Visual Navigation for Flying Robots

Lecture Notes
Summer Term 2013

Lecturer: Dr. Jürgen Sturm
Teaching Assistants: Jakob Engel, Christian Kerl

http://vision.in.tum.de/teaching/ss2013/visnav2013
Acknowledgements

This slide set would not have been possible without the help and support of many other people. In particular, I would like to thank all my great colleagues who made their lecture slides available for everybody on the internet or sent them to me personally.

My thanks go to (in alphabetical order)
- Alexander Kleiner
- Andrew Davison
- Andrew Zisserman
- Antonio Torralba
- Chad Jenkins
- Christian Kerl
- Cyrill Stachniss
- Daniel Cremers
- Frank Steinbrücker
- Friedrich Fraundorfer
- Georgio Grisetti
- Jan Peters
- Jakob Engel
- Jana Kosecka
- Jörg Müller
- Jürgen Hess
- Kai Arras
- Kai Wurm
- Korbinian Schmid
- Kurt Konolige
- Li Fei-Fei
- Maxim Likhachev
- Margaritha Chli
- Nicholas Roy
- Paul Newman
- Richard Newcombe
- Richard Szeliski
- Roland Siegwart
- Sebastian Thrun
- Steve Seitz
- Steven Lavalle
- Szymon Rusinkiewicz
- Volker Grabe
- Vijay Kumar
- Wolfram Burgard
Visual Navigation for Flying Robots

Welcome

Dr. Jürgen Sturm

Organization

- Tue 10:15-11:45
  - Lectures, discussions
  - Lecturer: Jürgen Sturm
- Thu 14:00-15:30
  - Lab course, homework & programming exercises
  - Teaching assistant: Jakob Engel and Christian Kerl

Who Are We?

- Computer Vision group:
  1 Professor, 3 Postdocs, 11 PhD students
- Research topics:
  Motion estimation, 3D reconstruction, image segmentation, convex optimization, shape analysis
- My research goal:
  Apply solutions from computer vision to real-world problems in robotics.
Who Are You?

Goal of this Course

- Provide an overview on problems/approaches for autonomous quadrocopters
- Strong focus on vision as the main sensor
- Areas covered: Mobile Robotics and Computer Vision
- Hands-on experience in lab course

Course Website

- Course Website: http://vision.in.tum.de/teaching/ss2013/visnav2013
  - Announcements
  - Schedule
  - Slides
  - Recordings
  - Exercises
- We need your feedback to improve this course!
- Let us know when you have ideas for improvement, find mistakes, ...

Course Material

  http://szeliski.org/Book/

Lecture Plan

1. Introduction
2. Robots, sensor and motion models
3. State estimation and control
4. Guest talks
5. Feature detection and matching
6. Motion estimation
7. Simultaneous localization and mapping
8. Stereo correspondence
9. 3D reconstruction
10. Navigation and path planning
11. Exploration
12. Evaluation and Benchmarking

Basics on mobile robotics
Camera-based localization and mapping
Advanced topics

Lab Course

- Jakob Engel and Christian Kerl
- Thu 14:00 – 15:30
- Room 02.05.014
- Alternation of exercises and robot lab:
  - Exercises: every two weeks, discussion of homework, participation is required
  - Robot lab: in weeks without exercises, help with quadrocopter programming, participation recommended
Exercises

- Exercise sheets contain both theoretical and programming problems
- 3 exercise sheets + 1 mini-project
- Deadline: before lecture (Tue 10:15)
- Hand in by email (visnav2013@vision.in.tum.de)

Group Assignment and Schedule

- 5 Parrot Ardrones
- 30 students in the course, 3 students per group → 10 groups
- Either use lab computers or bring own laptop (recommended)
- List for groups and robot schedule
- You have to sign up for a team before May 2nd (team list in lab room)
- After May 2nd, remaining places will be given to students on waiting list

Lab Course

- Starts this Thursday (room 02.05.014)
- Introduction to ROS and the Ardrone
- If you bring your own laptop:
  - Pre-install ROS
  - http://www.ros.org/wiki/ROS/Installation
- If not:
  - Jakob and Christian will provide you with user accounts for the lab machines

VISNAV2013: Team Assignment

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Student Name</th>
<th>Student Name</th>
<th>Student Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VISNAV2013: Robot Schedule

- Each team gets one time slot with programming support
- The robots/PCs are also available during the rest of the week (but without programming support)

<table>
<thead>
<tr>
<th>Thursday</th>
<th>Ardrone 1</th>
<th>Ardrone 2</th>
<th>Ardrone 3</th>
<th>Ardrone 4</th>
<th>Ardrone 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2pm – 4pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4pm – 6pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Safety Warning

- Quadrocopters are dangerous objects
- Read the instructions carefully before you start
- Always use the protective hull
- If somebody gets injured, report to us so that we can improve safety guidelines
- If something gets damaged, report it to us so that we can fix it
- NEVER TOUCH THE PROPELLORS
- DO NOT TRY TO CATCH THE QUADROCOPTER WHEN IT FAILS – LET IT FALL/CRAsh!
### General background
- Autonomous, automaton
  - self-willed (Greek, auto+matos)
- Robot
  - Karel Capek in 1923 play R.U.R. (Rossum’s Universal Robots)
  - labor (Czech or Polish, robota)
  - workman (Czech or Polish, robotnik)

### History
In 1966, Marvin Minsky at MIT asked his undergraduate student Gerald Jay Sussman to

“spend the summer linking a camera to a computer and getting the computer to describe what it saw”.

We now know that the problem is slightly more difficult than that. (Szeliski 2009, Computer Vision)

### Stanford Cart (1961-80)

![Stanford Cart](image)

Visual Mapping and Navigation by a Robot Rover
Stanford AI Lab
Thursday, October 27, 1999
Lucy Niche

### Shakey the Robot (1966-1972)

![Shakey the Robot](image)
Rhino and Minerva (1998-99)
- Museum tour guide robots
- University of Bonn and CMU
- Deutsches Museum, Smithsonian Museum

Roomba (2002)
- Sensor: one contact sensor
- Control: random movements
- Over 5 million units sold

Neato XV-11 (2010)
- Sensors:
  - 1D range sensor for mapping and localization
  - Improved coverage

Darpa Grand Challenge (2005)

Kiva Robotics (2007)
- Pick, pack and ship automation

Fork Lift Robots (2010)
**Quadrocopters (2001-)**

**Visual Navigation for Flying Robots**
Dr. Jürgen Sturm, Computer Vision Group, TUM

---

**Aggressive Maneuvers (2010)**

**Precise Aggressive Maneuvers for Autonomous Quadrotors**
Daniel Mellinger, Nathan Michael, Vijay Kumar
GRASP Lab, University of Pennsylvania

---

**Autonomous Construction (2011)**

**Construction with Quadrotor Teams**
Quentin Lindsey, Daniel Mellinger, Vijay Kumar
GRASP Lab, University of Pennsylvania

---

**Mapping with a Quadrocopter (2011)**

---

**Our Own Recent Work (2011-)**

- Visual odometry (Frank Steinbrücker, Christian Kerl)
- Camera-based navigation (Jakob Engel)
- 3D Reconstruction (Erik Bylow)

---

**Current Trends in Robotics**

Robots are entering novel domains

- Industrial automation
- Domestic service robots
- Medical, surgery
- Entertainment, toys
- Autonomous cars
- Aerial monitoring/inspection/construction
Flying Robots

- Recently increased interest in flying robots
- Shift focus to different problems (control is much more difficult for flying robots, path planning is simpler, ...) 
- Especially quadrocopters because
  - Can keep position
  - Reliable and compact
  - Low maintenance costs
- Trend towards miniaturization

Application Domains of Flying Robots

- Stunts for action movies, photography, sportscasts
- Search and rescue missions
- Aerial photogrammetry
- Documentation
- Aerial inspection of bridges, buildings, ...
- Construction tasks
- Military
- Today, quadrocopters are often still controlled by human pilots

Quadrocopter Platforms

- Commercial platforms
  - Ascending Technologies
  - Parrot Ardrone
  - ...
- Community/open-source projects
  - Mikrokopter
  - Paparazzi
  - ...

For more, see http://multicopter.org/wiki/Multicopter_Table

Flying Principles

- Fixed-wing airplanes
  - generate lift through forward airspeed and the shape of the wings
  - controlled by flaps
- Helicopters/rotorcrafts
  - main rotor for lift, tail rotor to compensate for torque
  - controlled by adjusting rotor pitch
- Quadrocopter/quadrotor
  - four rotors generate thrust
  - controlled by changing the speeds of rotation

Helicopter

- Swash plate adjusts pitch of propeller cyclically, controls pitch and roll
- Yaw is controlled by tail rotor

Quadrocopter

Keep position:
- Torques of all four rotors sum to zero
- Thrust compensates for earth gravity
Quadrocopter: Basic Motions

- Ascend
- Descend
- Turn Left
- Turn Right
- Accelerate Forward
- Accelerate to the Right
- Accelerate Backward
- Accelerate to the Left

Autonomous Flight
- Low level control (not covered in this course)
  - Maintain attitude, stabilize
  - Compensate for disturbances
- High level control
  - Compensate for drift
  - Avoid obstacles
  - Localization and Mapping
  - Navigate to point
  - Return to take-off position
  - Person following

Challenges
- Limited payload
- Limited computational power
- Limited sensors
- Limited battery life
- Fast dynamics, needs electronic stabilization
- Quadrocopter is always in motion
- Safety considerations
Robot Ethics

Three Laws of Robotics (Asimov, 1942):
- A robot may not injure a human being or, through inaction, allow a human being to come to harm.
- A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Robot Design

Imagine that we want to build a robot that has to perform navigation tasks...

How would you tackle this?
- What hardware would you choose?
- What software architecture would you choose?

Evolution of Paradigms in Robotics

- Classical robotics (mid-70s)
  - Exact models
  - No sensing necessary
- Reactive paradigms (mid-80s)
  - No models
  - Relies heavily on good sensing
- Hybrid approaches (since 90s)
  - Model-based at higher levels
  - Reactive at lower levels
- Current trends (since mid 2000s)
  - Big data
  - Cloud computing

Robot Hardware/Components

- Sensors
- Actuators
- Control Unit/Software

Classical / hierarchical paradigm

- Inspired by methods from Artificial Intelligence (70's)
- Focus on automated reasoning and knowledge representation
- STRIPS (Stanford Research Institute Problem Solver): Perfect world model, closed world assumption
- Shakey: Find boxes and move them to designated positions
**Classical paradigm: Stanford Cart**

- Take nine images of the environment, identify interesting points, estimate depth
- Integrate information into global world model
- Correlate images with previous image set to estimate robot motion
- On basis of desired motion, estimated motion, and current estimate of environment, determine direction in which to move
- Execute motion

**Characteristics of hierarchical paradigm**

Good old-fashioned Artificial Intelligence (GOFAI):

- Symbolic approaches
- Robot perceives the world, plans the next action, acts
- All data is inserted into a single, global world model
- Sequential data processing

**Reactive Paradigm**

- Sense-act type of organization
- Multiple instances of stimulus-response loops (called behaviors)
- Each behavior uses local sensing to generate the next action
- Combine several behaviors to solve complex tasks
- Run behaviors in parallel, behavior can override (subsume) output of other behaviors

**Characteristics of Reactive Paradigm**

- Situated agent, robot is integral part of the world
- No memory, controlled by what is happening in the world
- Tight coupling between perception and action via behaviors
- Only local, behavior-specific sensing is permitted (ego-centric representation)
Subsumption Architecture

- Introduced by Rodney Brooks in 1986
- Behaviors are networks of sensing and acting modules (augmented finite state machines)
- Modules are grouped into layers of competence
- Layers can subsume lower layers

Level 1: Avoid

Level 2: Wander

Level 3: Follow Corridor

Roomba Robot

Exercise: Model the behavior of a Roomba robot.

Navigation with Potential Fields

- Treat robot as a particle under the influence of a potential field
- Robot travels along the derivative of the potential
- Field depends on obstacles, desired travel directions and targets
- Resulting field (vector) is given by the summation of primitive fields
- Strength of field may change with distance to obstacle/target
Primitive Potential Fields

- Uniform
- Perpendicular
- Attractive
- Repulsive
- Tangential

Example: reach goal and avoid obstacles

Corridor Following Robot
- Level 1 (collision avoidance)
  add repulsive fields for the detected obstacles
- Level 2 (wander)
  add a uniform field into a (random) direction
- Level 3 (corridor following)
  replaces the wander field by three fields (two perpendicular, one parallel to the walls)

Characteristics of Potential Fields
- Simple method which is often used
- Easy to visualize
- Easy to combine different fields (with parameter tuning)
- But: Suffer from local minima
  Random motion to escape local minimum
  Backtracking
  Increase potential of visited regions
- High-level planner

Hybrid deliberative/reactive Paradigm

- Combines advantages of previous paradigms
  World model used in high-level planning
  Closed-loop, reactive low-level control

Modern Robot Architectures
- Robots became rather complex systems
- Often, a large set of individual capabilities is needed
- Flexible composition of different capabilities for different tasks
**Best Practices for Robot Architectures**

- Modular
- Robust
- De-centralized
- Facilitate software re-use
- Hardware and software abstraction
- Provide introspection
- Data logging and playback
- Easy to learn and to extend

**Robotic Middleware**

- Provides infrastructure
- Communication between modules
- Data logging facilities
- Tools for visualization
- Several systems available
  - Open-source: ROS (Robot Operating System), Player/Stage, CARMEN, YARP, OROCOS
  - Closed-source: Microsoft Robotics Studio

**Example Architecture for Navigation**

- **User interface / mission planning**
- **Global path planning**
  - **Localization module**
  - **Local path planning + collision avoidance**
- **Sensor interface(s)**
- **Actuator interface(s)**
- **Sensor driver(s)**
- **Actuator driver(s)**
- **Robot Hardware**

**Stanley’s Software Architecture**

- **Sensor Interface**
- **Perception**
- **Planning/Control**
- **User Interface**

**PR2 Software Architecture**

- Two 7-DOF arms, grippers, torso, 2-DOF head
- 7 cameras, 2 laser scanners
- Two 8-core CPUs, 3 network switches
- 73 nodes, 328 message topics, 174 services

**Communication Paradigms**

- Message-based communication
- Direct (shared) memory access
Forms of Communication

- Push
- Pull
- Publisher/subscriber
- Publish to blackboard
- Remote procedure calls / service calls
- Preemptive tasks / actions

Push

- Broadcast
- One-way communication
- Send as the information is generated by the producer P

Data delivery diagram:

```
P --> data --> C
```

Pull

- Data is delivered upon request by the consumer C (e.g., a map of the building)
- Useful if the consumer C controls the process and the data is not required (or available) at high frequency

Data delivery diagram:

```
P --> data request --> C
```

Publisher/Subscriber

- The consumer C requests a subscription for the data by the producer P (e.g., a camera or GPS)
- The producer P sends the subscribed data as it is generated to C
- Data generated according to a trigger (e.g., sensor data, computations, other messages, ...)

Data delivery diagram:

```
P --> subscription request --> C
```

Publish to Blackboard

- The producer P sends data to the blackboard (e.g., parameter server)
- A consumer C pull data from the blackboard B
- Only the last instance of data is stored in the blackboard B

Data delivery diagram:

```
P --> data request --> B --> data --> C
```

Service Calls

- The client C sends a request to the server S
- The server returns the result
- The client waits for the result (synchronous communication)
- Also called: Remote Procedure Call

Data delivery diagram:

```
C --> request + input data --> S --> result
```
**Actions (Preemptive Tasks)**

- The client requests the execution of an enduring action (e.g., navigate to a goal location)
- The server executes this action and sends continuously status updates
- Task execution may be canceled from both sides (e.g., timeout, new navigation goal,...)

**Robot Operating System (ROS)**

- We will use ROS in the lab course
- [http://www.ros.org/](http://www.ros.org/)
- Installation instructions, tutorials, docs

**Concepts in ROS**

- Nodes: programs that communicate with each other
- Messages: data structure (e.g., “Image”)
- Topics: typed message channels to which nodes can publish/subscribe (e.g., “/camera1/image_color”)
- Parameters: stored in a blackboard

**Software Management**

- Package: atomic unit of building, contains one or more nodes and/or message definitions
- Stack: atomic unit of releasing, contains several packages with a common theme
- Repository: contains several stacks, typically one repository per institution

**Useful Tools**

- `roscreate-pkg`
- `rosmake`
- `roscore`
- `rosnode list/info`
- `rostopic list/echo`
- `rosbag record/play`
- `rosrun`

**Tutorials in ROS**
Summary

- History of mobile robotics
- Brief intro on quadrocopters
- Paradigms in robotics
- Architectures and middleware

Questions?

- See you next week!

Organization: Lab Course

- Robot lab: room 02.05.14 (different room!)
- Lecture: room 02.09.23 (here)
- You have to sign up for a team before May 2\textsuperscript{nd} (team list in student lab)
- After May 2\textsuperscript{nd}, remaining places will be given to students on waiting list
- First exercise sheet is due next Tuesday 10am

Today’s Agenda

- Linear algebra
- 2D and 3D geometry
- Sensors
- First exercise sheet

Vectors

- Vector and its coordinates
  \[ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n \]

- Vectors represent points in an n-dimensional space
Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- Cross product

Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- Cross product

\[ \|x\|_2 = \|x\| = \sqrt{x_1^2 + x_2^2 + \ldots} \]

Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- Cross product

\[ \hat{x} = \frac{x}{\|x\|} \]

Vector Operations

- Scalar multiplication
- Addition/subtraction
- Length
- Normalized vector
- Dot product
- Cross product

\[ x \cdot y = \|x\| \|y\| \cos \theta \]

\[ x \times y = \|x\| \|y\| \sin(\theta) n \]

\( x, y \) are orthogonal if \( x \cdot y = 0 \)

\( y \) is lin. dependent from \( \{x_1, x_2, \ldots\} \) if \( y = \sum k_i x_i \)
Cross Product

- **Definition**
  \[ \mathbf{x} \times \mathbf{y} = \begin{pmatrix} x_2y_3 - x_3y_2 \\ x_3y_1 - x_1y_3 \\ x_1y_2 - x_2y_1 \end{pmatrix} \]

- **Matrix notation for the cross product**
  \[ [\mathbf{x}] \times = \begin{pmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{pmatrix} \]

- **Verify that** \( \mathbf{x} \times \mathbf{y} = [\mathbf{x}] \times \mathbf{y} \)

Matrices

- **Rectangular array of numbers**
  \[ X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix} \in \mathbb{R}^{n \times m} \]

- **First index refers to row**
- **Second index refers to column**

- **Column vectors of a matrix**
  \[ X = \begin{pmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{n1} \end{pmatrix} = (x_{11}, x_{21}, \ldots, x_{n1}) \]

- **Row vectors of a matrix**
  \[ X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix}^T = \begin{pmatrix} x_{11}^T \\ x_{21}^T \\ \vdots \\ x_{n1}^T \end{pmatrix} \]

  **Geometric interpretation:** for example, column vectors can form basis of a coordinate system

- **Square matrix**
- **Diagonal matrix**
- **Upper and lower triangular matrix**
- **Symmetric matrix**
- **Skew-symmetric matrix**
- **(Semi-)positive definite matrix**
- **Invertible matrix**
- **Orthonormal matrix**
- **Matrix rank**

- **Matrix rank**
  \[ a^T X a \geq 0 \]
Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication
- Inversion

Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- **Matrix-vector multiplication** $Xb$
- **Matrix-matrix multiplication**
- Inversion

Matrix-Vector Multiplication

- Geometric interpretation:
  A linear combination of the columns of $A$ scaled by the coefficients of $b$
  $\rightarrow$ coordinate transf. from local to global frame

Matrix-Matrix Multiplication

- Operator $\mathbb{R}^{n \times m} \times \mathbb{R}^{m \times p} \rightarrow \mathbb{R}^{n \times p}$
- Definition $C = AB = A \left( b_{1}, b_{2}, \ldots, b_{m} \right)$
- Interpretation: transformation of coordinate systems
- Can be used to concatenate transforms

Matrix-Matrix Multiplication

- Not commutative (in general) $AB \neq BA$
- Associative $A(BC) = (AB)C$
- Transpose $(AB)^{T} = B^{T}A^{T}$
Matrix Operations

- Scalar multiplication
- Addition/subtraction
- Transposition
- Matrix-vector multiplication
- Matrix-matrix multiplication
- Inversion

Matrix Inversion

- If \( A \) is a square matrix of full rank, then there is a unique matrix \( B = A^\top \) such that \( AB = I \).
- Different ways to compute, e.g., Gauss-Jordan elimination, LU decomposition, ...
- When \( A \) is orthonormal, then \( A^{-1} = A^\top \)

Recap: Linear Algebra

- Vectors
- Matrices
- Operators

Now let’s apply these concepts to 2D+3D geometry

Geometric Primitives in 2D

- 2D point \( x = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2 \)
- Augmented vector \( \tilde{x} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in \mathbb{R}^3 \)
- Homogeneous coordinates \( \tilde{x} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^2 \)

Geometric Primitives in 2D

- Homogeneous vectors that differ only be scale represent the same 2D point
- Convert back to inhomogeneous coordinates by dividing through last element
  \( \tilde{x} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} x/\tilde{w} \\ y/\tilde{w} \\ 1 \end{pmatrix} = \tilde{w} \tilde{x} \)
- Points with \( \tilde{w} = 0 \) are called points at infinity or ideal points

- 2D line \( \tilde{l} = (a, b, c)^\top \)
- 2D line equation \( \tilde{x} \cdot \tilde{l} = ax + by + c = 0 \)

Visual Navigation for Flying Robots
Geometric Primitives in 2D

- Normalized line equation vector
  \[ \vec{I} = (\hat{n}_x, \hat{n}_y, d)^\top = (\hat{n}, d)^\top \quad \text{with} \quad \|\hat{n}\| = 1 \]
  where \( d \) is the distance of the line to the origin

Geometric Primitives in 2D

- Polar coordinates of a line: \((\theta, d)^\top\)
  (e.g., used in Hough transform for finding lines)
  \[ \hat{n} = (\cos \theta, \sin \theta)^\top \]

Geometric Primitives in 2D

- Line joining two points
  \[ \vec{I} = \vec{x}_1 \times \vec{x}_2 \]

- Intersection point of two lines
  \[ \vec{x} = \vec{I}_1 \times \vec{I}_2 \]

Geometric Primitives in 3D

- 3D point
  \[ \vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3 \]

- Augmented vector
  \[ \vec{x} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in \mathbb{R}^4 \]

- Homogeneous coordinates
  \[ \vec{x} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^3 \]

Geometric Primitives in 3D

- 3D plane
  \[ \hat{\vec{m}} = (a, b, c, d)^\top \]

- 3D plane equation
  \[ \vec{x} \cdot \hat{\vec{m}} = ax + by + cz + d = 0 \]

- Normalized plane with unit normal vector
  \[ \vec{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d)^\top = (\hat{n}, d) \]
  \((\|\hat{n}\| = 1)\)
  and distance \( d \)

Geometric Primitives in 3D

- 3D line
  \[ \vec{r} = (1 - \lambda)\vec{p} + \lambda \vec{q} \]
  through points \( \vec{p}, \vec{q} \)

- Infinite line:
  \[ \lambda \in \mathbb{R} \]

- Line segment joining \( \vec{p}, \vec{q} \)
  \[ 0 \leq \lambda \leq 1 \]
2D Planar Transformations

2D Transformations

- Translation

\[ x' = x + t \]
\[ \bar{x}' = \begin{pmatrix} 1 & t \\ \bar{x} \end{pmatrix} \]
\[ \bar{x}' = \begin{pmatrix} 1 & 0^T \\ 0^T & 1 \end{pmatrix} \bar{x} \]

where \( t \in \mathbb{R}^2 \) is the translation vector, \( I \) is the identity matrix, and \( 0 \) is the zero vector.

Rigid body motion or Euclidean transformation (rotation + translation)

\[ x' = Rx + t \quad \text{or} \quad \bar{x}' = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \bar{x} \]

where \( R = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \) is an orthonormal rotation matrix, i.e., \( RR^T = I \).

Distances (and angles) are preserved.

Scaled rotation/similarity transform

\[ x' = sRx + t \quad \text{or} \quad \bar{x}' = \begin{pmatrix} sR \\ 0^T \end{pmatrix} \bar{x} \]

Preserves angles between lines.

Affine transform

\[ x' = Ax = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \bar{x} \]

Parallel lines remain parallel.
2D Transformations

- Projective/perspective transform
  \[ \mathbf{\tilde{x}'} = \mathbf{\tilde{H}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \mathbf{\tilde{x}} \]

- Note that \( \mathbf{\tilde{H}} \) is homogeneous (only defined up to scale)
- Resulting coordinates are homogeneous
- Lines remain lines :-)

### Examples: Euclidean Transformations

- Robot is located somewhere in space

\[ X = \begin{pmatrix} R \\ t \end{pmatrix} = \begin{pmatrix} \cos \psi & -\sin \psi & x \\ \sin \psi & \cos \psi & y \\ 0 & 0 & 1 \end{pmatrix} \in \text{SE}(2) \subset \mathbb{R}^{3\times3} \]

### Coordinate Transforms

- Robot is located somewhere in space

\[ X = \begin{pmatrix} \cos 45 & -\sin 45 & 0.7 \\ \sin 45 & \cos 45 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \]

### Coordinate Transforms

- Robot is located at \( x=0.7, \ y=0.5, \ \text{yaw}=45\text{deg} \)

\[ X = \begin{pmatrix} \cos 45 & -\sin 45 & 0.7 \\ \sin 45 & \cos 45 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \]
Vector Transformation

- Robot is located at $x=0.7$, $y=0.5$, $yaw=45$deg
- Robot moves 1m forward
- What is its position afterwards?

$$\begin{align*}
\vec{v}_{\text{global}} &= X \vec{v}_{\text{local}} \\
&= \begin{pmatrix}
0.71 & -0.71 & 0.7\\
0.71 & 0.71 & 0.5\\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 \\
0 \\
1
\end{pmatrix} \\
&= \begin{pmatrix}
1.41 \\
1.21 \\
1
\end{pmatrix}
\end{align*}$$

Inhomogeneous coordinates

Homogeneous coordinates
Vector Transformation

- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\mathbf{v}_{\text{global}} = X \mathbf{v}_{\text{local}}$$

Inverse Transformations

- We transformed local to global coordinates
- Sometimes we need to do the inverse
- How can we transform global coordinates into local coordinates?

$$\mathbf{v}_{\text{global}} = X \mathbf{v}_{\text{local}}$$

$$\mathbf{v}_{\text{local}} = X^{-1} \mathbf{v}_{\text{global}}$$

Coordinate System Transformations

- Robot moves 0.2m forward, 0.1m sideways, and turns by 10deg

$$U_1 = \begin{pmatrix} \cos 10 & -\sin 10 & 0.2 \\ \sin 10 & \cos 10 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix}$$

Coordinate System Transformations

- After this motion, the robot pose (in the global frame) becomes

$$X_2 = XU$$

$$= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \cdots$$
Coordinate System Transformations

Note: The order matters
- Move 1m forward, then turn 90deg left
- Turn 90deg left, then move 1m forward

\[ AB \neq BA \]

3D Transformations

- Translation

\[
\mathbf{x}' = \begin{pmatrix} 1 & t \\ 0 & 1 \end{pmatrix} \mathbf{x}
\]

- Euclidean transform (translation + rotation), (also called the Special Euclidean group SE(3))

\[
\mathbf{x}' = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \mathbf{x}
\]

- Scaled rotation, affine transform, projective transform...

3D Euclidean Transformations

- Translation \( t \) has 3 degrees of freedom
- Rotation \( R \) has 3 degrees of freedom

\[
X = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}
\]

3D Rotations

- Rotation matrix
  (also called the special orientation group SO(3))

- Euler angles
- Axis/angle
- Unit quaternion

Rotation Matrix

- Orthonormal 3x3 matrix

\[
R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}
\]

- Column vectors correspond to coordinate axes
- Special orientation group \( R \in SO(3) \)
- What operations do we typically do with rotation matrices?
Rotation Matrix

- Orthonormal 3x3 matrix

\[
R = \begin{pmatrix}
    r_{11} & r_{12} & r_{13} \\
    r_{21} & r_{22} & r_{23} \\
    r_{31} & r_{32} & r_{33}
\end{pmatrix}
\]

- Advantage: Can be easily concatenated and inverted (how?)
- Disadvantage: Over-parameterized (9 parameters instead of 3)

Euler Angles

- Product of 3 consecutive rotations (e.g., around X-Y-Z axes)
- Roll-pitch-yaw convention is very common in aerial navigation (DIN 9300)

Roll-Pitch-Yaw Convention

- Yaw \( \psi \) Pitch \( \theta \) Roll \( \phi \) to rotation matrix

\[
R = R_Z(\phi)R_Y(\theta)R_X(\psi)
\]

\[
= \begin{pmatrix}
    1 & 0 & 0 \\
    0 & \cos \phi & \sin \phi \\
    0 & -\sin \phi & \cos \phi
\end{pmatrix}
\begin{pmatrix}
    \cos \theta & 0 & -\sin \theta \\
    0 & 1 & 0 \\
    \sin \theta & 0 & \cos \theta
\end{pmatrix}
\begin{pmatrix}
    \cos \psi & \sin \psi & 0 \\
    -\sin \psi & \cos \psi & 0 \\
    0 & 0 & 1
\end{pmatrix}
\]

- Rotation matrix to Yaw-Pitch-Roll

\[
\phi = \text{Atan2} \left( -r_{31}, \sqrt{r_{11}^2 + r_{21}^2} \right)
\]

\[
\psi = -\text{Atan2} \left( \frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)} \right)
\]

\[
\theta = \text{Atan2} \left( \frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)} \right)
\]

Euler Angles

- Advantage:
  - Minimal representation (3 parameters)
  - Easy interpretation
- Disadvantages:
  - Many “alternative” Euler representations exist (XYZ, ZXZ, ZYX, ...)
  - Difficult to concatenate
  - Singularities (gimbal lock)

Gimbal Lock

- When the axes align, one degree-of-freedom (DOF) is lost...
**Axis/Angle**

- Represent rotation by
  - rotation axis \( \hat{n} \) and
  - rotation angle \( \theta \)
- 4 parameters \((\hat{n}, \theta)\)
- 3 parameters \(\omega = \theta \hat{n}\)
  - length is rotation angle
  - also called the angular velocity
  - minimal but not unique (why?)

**Conversion**

- Rodriguez’ formula
  \[
  R(\hat{n}, \theta) = I + \sin \theta [\hat{n}]_x + (1 - \cos \theta)[\hat{n}]_x^2
  \]
- Inverse
  \[
  \theta = \cos^{-1}
  \left(\frac{\text{trace}(R) - 1}{2}\right), \hat{n} = \frac{1}{2\sin\theta}
  \left(\begin{array}{c}
  r_{32} - r_{23} \\
  r_{13} - r_{31} \\
  r_{21} - r_{12}
  \end{array}\right)
  \]

  see: An Invitation to 3D Vision, Y. Ma, S. Soatto, J. Kosecka, S. Sastry, Chapter 2 (available online)

**Unit Quaternions**

- Quaternion \( q = (q_x, q_y, q_z, q_w)^T \in \mathbb{R}^4 \)
- Unit quaternions have \( \|q\| = 1 \)
- Opposite sign quaternions represent the same rotation \( q = -q \)
- Otherwise unique

**Unit Quaternions**

- Advantage: multiplication and inversion operations are efficient
- Quaternion-Quaternion Multiplication
  \[
  q_0 q_1 = (v_0, w_0)(v_1, w_1)
  = (v_0 \times v_1 + w_0 v_1 + w_1 v_0, w_0 w_1 - v_0 v_1)
  \]
- Inverse (flip sign of \( v \) or \( w \))
  \[
  q^{-1} = (v, w)^{-1}
  = (v, -w)
  \]

**Unit Quaternions**

- Quaternion-Vector multiplication (rotate point \( p \) with rotation \( q \))
  \[
  p' = v p q^{-1}
  \]
  with \( \vec{p} = (x, y, z, 0)^T \)
- Relation to Axis/Angle representation
  \[
  q = (v, w) = (\sin \frac{\theta}{2} \hat{n}, \cos \frac{\theta}{2})
  \]

**3D Orientations**

- **Note:** In general, it is very hard to “read” 3D orientations/rotations, no matter in what representation
- **Observation:** They are usually easy to visualize and can then be intuitively interpreted
- **Advice:** Use 3D visualization tools for debugging (RVIZ, libqglviewer, ...)

Visual Navigation for Flying Robots  71  Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots  72  Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots  73  Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots  74  Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots  75  Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots  76  Dr. Jürgen Sturm, Computer Vision Group, TUM
C++ Libraries for Lin. Alg./Geometry

- Many C++ libraries exist for linear algebra and 3D geometry
- Typically conversion necessary
- Examples:
  - C arrays, std::vector (no linear alg. functions)
  - gsl (gnu scientific library, many functions, plain C)
  - boost::array (used by ROS messages)
  - Bullet library (3D geometry, used by ROS tf)
  - Eigen (both linear algebra and geometry, my recommendation)

Example: Transform Trees in ROS

- TF package represents 3D transforms between rigid bodies in the scene as a tree
- Collects transformations
- Simple query interface

Example: Video from PR2

3D to 2D Projections

- Orthographic projections
- Perspective projections

3D to 2D Perspective Projection
3D to 2D Perspective Projection

- 3D point $\mathbf{p}$ (in the camera frame)
- 2D point $\mathbf{x}$ (on the image plane)
- Pin-hole camera model

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \mathbf{p}$$

- Remember, $\mathbf{X}$ is homogeneous, need to normalize

$$\mathbf{X} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \Rightarrow \mathbf{x} = \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix}$$

Camera Intrinsics

- So far, 2D point is given in meters on image plane
- But: we want 2D point be measured in pixels (as the sensor does)

Camera Extrinsic

- Assume $\mathbf{p}_w$ is given in world coordinates
- Transform from world to camera (also called the camera extrinsics)

$$\mathbf{p} = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \mathbf{p}_w$$

Recap: 2D/3D Geometry

- Points, lines, planes
- 2D and 3D transformations
- Different representations for 3D orientations
- Choice depends on application
- Which representations do you remember?
- 3D to 2D perspective projections

- You really have to know 2D/3D transformations by heart (read Szeliski, Chapter 2)

Sensors
**Sensors**
- Tactile sensors
  - Contact switches, bumpers, proximity sensors, pressure
- Wheel/motor sensors
  - Potentiometers, brush/optical/magnetic/inductive/capacitive encoders, current sensors
- Heading sensors
  - Compass, infrared, inclinometers, gyroscopes, accelerometers
- Ground-based beacons
  - GPS, optical or RF beacons, reflective beacons
- Active ranging
  - Ultrasonic sensor, laser rangefinder, optical triangulation, structured light
- Motion/speed sensors
  - Doppler radar, Doppler sound
- Vision-based sensors
  - CCD/CMOS cameras, visual servoing packages, object tracking packages

**Example: Ardrone Sensors**
- Tactile sensors
  - Contact switches, bumpers, proximity sensors, pressure
- Wheel/motor sensors
  - Potentiometers, brush/optical/magnetic/inductive/capacitive encoders, current sensors
- Heading sensors
  - Compass, infrared, inclinometers, gyroscopes, accelerometers
- Ground-based beacons
  - GPS, optical or RF beacons, reflective beacons
- Active ranging
  - Ultrasonic sensor, laser rangefinder, optical triangulation, structured light
- Motion/speed sensors
  - Doppler radar, Doppler sound
- Vision-based sensors
  - CCD/CMOS cameras, visual servoing packages, object tracking packages

**Characterization of Sensor Performance**
- Bandwidth or Frequency
- Delay
- Sensitivity
- Cross-sensitivity (cross-talk)
- Error (accuracy)
  - Deterministic errors (modeling/calibration possible)
  - Random errors
- Weight, power consumption, ...

**Let’s Have a Closer Look**
- Cameras
- Gyroscope
- Accelerometers
- GPS
- Range sensors

**Pinhole Camera**
- Lit scene emits light
- Film/sensor is light sensitive

**Lens Camera**
- Lit scene emits light
- Film/sensor is light sensitive
- A lens focuses rays onto the film/sensor
Real Cameras

- Radial distortion of the image
- Caused by imperfect lenses
- Deviations are most noticeable for rays that pass through the edge of the lens

Radial Distortion

- Radial distortion of the image
- Caused by imperfect lenses
- Deviations are most noticeable for rays that pass through the edge of the lens
- Typically compensated with a low-order polynomial
  \[
  \hat{x}_c = x_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)
  \]
  \[
  \hat{y}_c = y_c(1 + \kappa_1 r_c^2 + \kappa_2 r_c^4)
  \]

Digital Cameras

- Vignetting
- De-bayering
- Rolling shutter and motion blur
- Compression (JPG)
- Noise

Mechanical Gyroscope

- Measures orientation (standard gyro) or angular velocity (rate gyro, needs integration for angle)
- Spinning wheel mounted in a gimbal device (can move freely in 3 dimensions)
- Wheel keeps orientation due to angular momentum (standard gyro)

Modern Gyroscopes

- Vibrating structure gyroscope (MEMS)
  - Based on Coriolis effect
  - "Vibration keeps its direction under rotation"
  - Implementations: Tuning fork, vibrating wheels, ...
- Ring laser / fibre optic gyro
  - Interference between counter-propagating beams in response to rotation

Accelerometer

- Measures all external forces acting upon them (including gravity)
- Acts like a spring-damper system
- To obtain inertial acceleration (due to motion alone), gravity must be subtracted
MEMS Accelerometers
- Micro Electro-Mechanical Systems (MEMS)
- Spring-like structure with a proof mass
- Damping results from residual gas
- Implementations: capacitive, piezoelectric, ...

Inertial Measurement Unit
- 3-axes MEMS gyroscope
  - Provides angular velocity
  - Integrate for angular position
  - Problem: Drifts slowly over time (e.g., 1deg/hour), called the bias
- 3-axes MEMS accelerometer
  - Provides accelerations (including gravity)
  - Can we use these sensors to estimate our position?

Example: AscTec Autopilot Board

GPS
- 24+ satellites, 12 hour orbit, 20.190 km height
- 6 orbital planes, 4+ satellites per orbit, 60deg distance
- Satellite transmits orbital location + time
- 50bits/s, msg has 1500 bits → 12.5 minutes
Range Sensors

- Sonar
- Laser range finder
- Time of flight camera
- Structured light (will be covered later)

Ultrasonic Range Sensors

- Range between 12cm and 5m
- Opening angle around 20 to 40 degrees
- Soft surfaces absorb sound
- Reflections → ghosts
- Lightweight and cheap

Laser Scanner

- Measures phase shift
- Pro: High precision, wide field of view, safety approved for collision detection
- Con: Relatively expensive + heavy

Exercise Sheet 1
The pose of a robot can be described by 6 parameters:

- Three-dimensional Cartesian coordinates
- Three Euler angles roll, pitch, yaw.

The state space of such a system is six-dimensional: $$x_t = (x, y, z, \phi, \theta, \psi)^T$$

Robot makes sensor observations usually in its ego-centric frame (as seen by the robot).

In practice, one often finds two types of motion models:

- Odometry-based
- Velocity-based (dead reckoning)

Odometry-based models are used when systems are equipped with distance sensors (e.g., wheel encoders).

Velocity-based models have to be applied when no wheel encoders are given.

Exercise Sheet 1:

- Odometry sensor on Ardrone is an integrated package
- Sensors
  - Down-looking camera to estimate motion
  - Ultrasonic sensor to get height
  - 3-axes gyroscopes
  - 3-axes accelerometer
- IMU readings $$u_t$$ (in provided bag file)
  - Horizontal speed ($v_x/v_y$) in its local frame (l)
  - Height ($z$) in the global frame
  - Roll, Pitch, Yaw in the global frame
- Integrate these values to get robot pose $$x_t = f(x_{t-1}, u_t)$$
  - Position ($x/y/z$) in the global frame
  - Orientation (e.g., $r/p/y$) in the global frame

Due next Tuesday, 10am
Hand in via email to visnav2013@vision.in.tum.de
Visual Navigation for Flying Robots
Probabilistic Models and State Estimation

Dr. Jürgen Sturm

Scientific Research

Be Creative & Do Research

Write and Submit Paper

Present Work at Conference

Prepare Talk/Poster

Flow of Publications

Research

MSc Thesis

Conference Paper

Conference Paper

Conference Paper

Conference Paper

Journal Article

Journal Article

PhD Thesis

ICRA, IROS, CVPR, ICCV, NIPS, ...

T-RO, IJC, RAS, PAMI, ...

Important Conferences

Robotics
- International Conference on Robotics and Automation (ICRA) next week
- International Conference on Intelligent Robots and Systems (IROS)
- Robotics: Science and Systems (RSS)

Computer Vision
- International Conference on Computer Vision (ICCV)
- Computer Vision and Pattern Recognition (CVPR)
- German Conference on Computer Vision (GCPR)

ICRA: Preview
- Christian’s work got nominated for the “Best Vision Paper”
- Four sessions (with 6 papers each) on flying robots
- Interesting sessions:
  - Localization
  - Theory and Methods for SLAM
  - Visual Servoing
  - Sensor Fusion
  - Trajectory Planning for Aerial Robots
  - Novel Aerial Robots
  - Aerial Robots and Manipulation
  - Modeling & Control of Aerial Robots
  - Sensing for Navigation
  - ...
- Will report on interesting papers!
**Perception**

- Perception and models are strongly linked

**Example: Human Perception**

- Count the black dots

**Models in Human Perception**

- Count the black dots

**State Estimation**

- Cannot observe world state directly
- Need to estimate the world state
- Robot maintains belief about world state
- Update belief according to observations and actions using models
- Sensor observations → sensor model
- Executed actions → action/motion model

more on [http://michaelbach.de/ot/index.html](http://michaelbach.de/ot/index.html)
State Estimation

What parts of the world state are (most) relevant for a flying robot?

- Position
- Velocity
- Obstacles
- Map
- Positions and intentions of other robots/humans
- ...

Models and State Estimation

(Deterministic) Sensor Model

- Robot perceives the environment through its sensors
  \[ z = h(x) \]
  \[ z \]  
  \[ h \]
  \[ x \]
  \[ \]  

(Deterministic) Motion Model

- Robot executes an action \( u \) (e.g., move forward at 1m/s)

- Update belief state according to motion model
  \[ x' = g(x, u) \]

Probabilistic Robotics

- Sensor observations are noisy, partial, potentially missing (why?)
- All models are partially wrong and incomplete (why?)
- Usually we have prior knowledge (from where?)
Probabilistic Robotics

- Probabilistic sensor and motion models
  \( p(z \mid x) \) \( p(x' \mid x, u) \)
- Integrate information from multiple sensors (multi-modal)
  \( p(x \mid z_{\text{vision}}, z_{\text{ultrasound}}, z_{\text{IMU}}) \)
- Integrate information over time (filtering)
  \( p(x \mid z_1, z_2, \ldots, z_t) \)
  \( p(x \mid u_1, z_1, \ldots, u_t, z_t) \)

Agenda for Today

- Motivation ✓
- Bayesian Probability Theory
- Bayes Filter
- Normal Distribution
- Kalman Filter

The Axioms of Probability Theory

Notation: \( P(A) \) refers to the probability that proposition \( A \) holds

1. \( 0 \leq P(A) \leq 1 \)
2. \( P(\Omega) = 1 \) \( P(\emptyset) = 0 \)
3. \( P(A \cup B) = P(A) + P(B) - P(A \cap B) \)

Discrete Random Variables

- \( X \) denotes a random variable
- \( X \) can take on a countable number of values in \( \{x_1, x_2, \ldots, x_n\} \)
- \( P(X = x_i) \) is the probability that the random variable \( X \) takes on value \( x_i \)
- \( P(\cdot) \) is called the probability mass function

Example: \( P(\text{Room}) = <0.7, 0.2, 0.08, 0.02> \)
Room \( \in \{\text{office, corridor, lab, kitchen}\} \)

Continuous Random Variables

- \( X \) takes on continuous values
- \( p(X = x) \) or \( p(x) \) is called the probability density function (PDF)
  \[ P(x \in [a, b]) = \int_a^b p(x) \, dx \]
- Example

\( p(x) \)
Proper Distributions Sum To One
- Discrete case: \( \sum_x P(x) = 1 \)
- Continuous case: \( \int p(x) \, dx = 1 \)

Joint and Conditional Probabilities
- \( P(X = x \text{ and } Y = y) = P(x, y) \)
- If \( X \) and \( Y \) are independent then
  \( P(x, y) = P(x)P(y) \)
  \( P(x \mid y)P(y) = P(x, y) \)
- If \( X \) and \( Y \) are independent then
  \( P(x \mid y) = P(x) \)

Conditional Independence
- Definition of conditional independence
  \( P(x, y \mid z) = P(x \mid z)P(y \mid z) \)
- Equivalent to
  \( P(x \mid z) = P(x \mid y, z) \)
  \( P(y \mid z) = P(x \mid x, z) \)
- Note: this does not necessarily mean that
  \( P(x, y) = P(x)P(y) \)

Marginalization
- Discrete case: \( P(x) = \sum_y P(x, y) \)
- Continuous case: \( p(x) = \int p(x, y) \, dy \)

Example: Marginalization
- \( P_{Y \mid X} \)
<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( P_{Y \mid X} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y_1 )</td>
<td>( \frac{1}{3} )</td>
<td>( \frac{1}{3} )</td>
<td>( \frac{1}{3} )</td>
<td>( \frac{1}{3} )</td>
</tr>
<tr>
<td>( y_2 )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
</tr>
<tr>
<td>( y_3 )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
</tr>
<tr>
<td>( y_4 )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
<td>( \frac{1}{6} )</td>
</tr>
</tbody>
</table>

Law of Total Probability
- Discrete case: \( P(x) = \sum_y P(x, y) \)
  \( = \sum_y P(x \mid y)P(y) \)
- Continuous case: \( p(x) = \int p(x, y) \, dy \)
  \( = \int p(x \mid y)p(y) \, dy \)
Expected Value of a Random Variable

- **Discrete case**
  \[ E[X] = \sum_i x_i P(x_i) \]

- **Continuous case**
  \[ E[X] = \int x P(X = x) \, dx \]

- The expected value is the weighted average of all values a random variable can take on.
- Expectation is a linear operator
  \[ E[aX + b] = aE[X] + b \]

Covariance of a Random Variable

- Measures the squared expected deviation from the mean
  \[ \text{Cov}[X] = E[X - E[X]]^2 = E[X^2] - E[X]^2 \]

Estimation from Data

- **Observations** \( x_1, x_2, \ldots, x_n \in \mathbb{R}^d \)
- **Sample Mean** \( \mu = \frac{1}{n} \sum_i x_i \)
- **Sample Covariance**
  \[ \Sigma = \frac{1}{n-1} \sum_i (x_i - \mu)(x_i - \mu)^T \]

The State Estimation Problem

We want to estimate the world state \( x \) from
1. Sensor measurements \( z \) and
2. Controls (or odometry readings) \( u \)

We need to model the relationship between these random variables, i.e.,
\[ p(x \mid z) \quad p(x' \mid x, u) \]

Causal vs. Diagnostic Reasoning

- \( P(x \mid z) \) is diagnostic
- \( P(z \mid x) \) is causal
- Often causal knowledge is easier to obtain
- Bayes rule allows us to use causal knowledge:
  \[
  P(x \mid z) = \frac{P(z \mid x)P(x)}{P(z)}
  \]
  \[
  \text{prior on world state} \quad \downarrow \quad \text{observation likelihood} \quad \uparrow \quad \text{prior on sensor observations}
  \]

Bayes Formula

- **Derivation of Bayes Formula**
  \[ P(x, z) = P(x \mid z)P(z) = P(z \mid x)P(x) \]
- **Our usage**
  \[
  P(x \mid z) = \frac{P(z \mid x)P(x)}{P(z)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}}
  \]
Bayes Formula

- Derivation of Bayes Formula
  \[ P(x, z) = P(x \mid z)P(z) = P(z \mid x)P(x) \]
- Our usage
  \[ P(x \mid z) = \frac{P(z \mid x)P(x)}{P(z)} = \frac{\text{likelihood} \cdot \text{prior}}{\text{evidence}} \]

Normalization

- Direct computation of \( P(z) \) can be difficult
- Idea: Compute improper distribution, normalize afterwards
  \[ \text{Step 1: } L(x \mid z) = P(z \mid x)P(x) \]
  \[ \text{Step 2: } P(z) = \sum_x P(z \mid x)P(x) = \sum_x L(x \mid z) \]
  \[ \text{Step 3: } P(x \mid z) = \frac{L(x \mid z)}{P(z)} \]

Bayes Rule with Background Knowledge

\[ P(x \mid y, z) = \frac{P(y \mid x, z)P(x \mid z)}{P(y \mid z)} \]

Example: Sensor Measurement

- Binary sensor \( Z \in \{\text{bright, } \neg\text{bright}\} \)
- Binary world state \( X \in \{\text{home, } \neg\text{home}\} \)
- Sensor model \( P(Z = \text{bright} \mid X = \text{home}) = 0.6 \)
  \( P(Z = \text{bright} \mid X = \neg\text{home}) = 0.3 \)
- Prior on world state \( P(X = \text{home}) = 0.5 \)
- Assume: Robot observes light, i.e., \( Z = \text{bright} \)
- What is the probability \( P(X = \text{home} \mid Z = \text{bright}) \) that the robot is above the landing zone?

Example: Sensor Measurement

- Sensor model \( P(Z = \text{bright} \mid X = \text{home}) = 0.6 \)
  \( P(Z = \text{bright} \mid X = \neg\text{home}) = 0.3 \)
- Prior on world state \( P(X = \text{home}) = 0.5 \)
- Probability after observation (using Bayes)
  \[ P(X = \text{home} \mid Z = \text{bright}) = \frac{P(\text{bright} \mid \text{home})P(\text{home})}{0.6 \cdot 0.5 + 0.3 \cdot 0.5} = \frac{0.6 \cdot 0.3}{0.3 + 0.15} = 0.67 \]
Example: Sensor Measurement

- **Sensor model** \( P(Z = \text{bright} \mid X = \text{home}) = 0.6 \)
  \( P(Z = \text{bright} \mid X = \neg \text{home}) = 0.3 \)
- **Prior on world state** \( P(X = \text{home}) = 0.5 \)
- **Probability after observation (using Bayes)**

\[
P(X = \text{home} \mid Z = \text{bright}) = \frac{P(\text{bright} \mid \text{home})P(\text{home})}{P(\text{bright} \mid \text{home})P(\text{home}) + P(\text{bright} \mid \neg \text{home})P(\neg \text{home})} = \frac{0.6 \cdot 0.5 + 0.3 \cdot 0.5}{0.3 + 0.15} = 0.67
\]

Combining Evidence

- Suppose our robot obtains another observation \( z_2 \) (either from the same or a different sensor)
- How can we integrate this new information?
- More generally, how can we estimate \( p(x \mid z_1, z_2, \ldots) \)?

Bayes formula gives us:

\[
P(x \mid z_1, \ldots, z_n) = \frac{P(z_n \mid x, z_1, \ldots, z_{n-1})P(x \mid z_1, \ldots, z_{n-1})}{P(z_n \mid z_1, \ldots, z_{n-1})}
\]

Recursive Bayesian Updates

- **Markov Assumption:** \( z_n \) is independent of \( z_1, \ldots, z_{n-1} \) if we know \( x \)
- \[
P(x \mid z_1, \ldots, z_n) = \frac{P(z_n \mid x, z_1, \ldots, z_{n-1})P(x \mid z_1, \ldots, z_{n-1})}{P(z_n \mid z_1, \ldots, z_{n-1})}
\]
- **Markov Assumption:**

\[
\Rightarrow P(x \mid z_1, \ldots, z_n) = \frac{P(z_n \mid x)P(x \mid z_1, \ldots, z_{n-1})}{P(z_n \mid z_1, \ldots, z_{n-1})}
\]

\[
= \eta P(x \mid z_n)P(x \mid z_1, \ldots, z_{n-1})
\]

\[
= \eta \prod_{i=1}^{n} P(z_i \mid x)P(x)
\]
Example: Sensor Measurement

- Quadrocopter seeks the landing zone
- Landing zone is marked with many bright lamps and a visual marker

Example: Second Measurement

- Sensor model \( P(Z_2 = \text{marker} \mid X = \text{home}) = 0.8 \)
  \( P(Z_2 = \text{marker} \mid X = \neg\text{home}) = 0.1 \)
- Previous estimate \( P(X = \text{home} \mid Z_1 = \text{bright}) = 0.67 \)
- Assume robot does not observe marker
- What is the probability of being home?

\[
P(X = \text{home} \mid Z_1 = \text{bright}, Z_2 = \neg\text{marker}) = \frac{P(Z_2 = \neg\text{marker} \mid \text{home})P(\text{home} \mid \text{bright})}{0.2 \cdot 0.67 + 0.9 \cdot 0.33} = 0.31
\]

The second observation lowers the probability that the robot is above the landing zone!

Actions (Motions)

- Often the world is dynamic since
  - actions carried out by the robot...
  - actions carried out by other agents...
  - or just time passing by...
  - ...change the world
- How can we incorporate actions?

Typical Actions

- Quadrocopter accelerates by changing the speed of its motors
- Position also changes when quadrocopter does “nothing” (and drifts..)
- Actions are never carried out with absolute certainty
- In contrast to measurements, actions generally increase the uncertainty of the state estimate

Action Models

- To incorporate the outcome of an action \( u \) into the current state estimate (“belief”), we use the conditional pdf

\[
p(x' \mid u, x)
\]

This term specifies the probability that executing the action \( u \) in state \( x \) will lead to state \( x' \)
Example: Take-Off

- Action: \( u \in \{\text{takeoff}\} \)
- World state: \( x \in \{\text{ground, air}\} \)

\[
P(x' \mid u) = \sum_{x} P(x' \mid u, x)P(x)
\]

Continuous case

\[
p(x' \mid u) = \int p(x' \mid u, x)p(x)dx
\]

Integrating the Outcome of Actions

Markov Chain

A Markov chain is a stochastic process where, given the present state, the past and the future states are independent

Bayes Filter

Given:
- Stream of observations \( z \) and actions \( u \):
  \( d_t = (u_1, z_1, \ldots, u_t, z_t)^T \)
- Sensor model \( P(z \mid x) \)
- Action model \( P(x' \mid x, u) \)
- Prior probability of the system state \( P(x) \)

Wanted:
- Estimate of the state \( x \) of the dynamic system
- Posterior of the state is also called belief

\[
Bel(x_t) = P(x_t \mid u_1, z_1, \ldots, u_t, z_t)
\]
Bayes Filter

For each time step, do
1. Apply motion model
\[
\text{Bel}(x_t) = \sum_{x_{t-1}} P(x_t \mid x_{t-1}, u_t) \text{Bel}(x_{t-1})
\]
2. Apply sensor model
\[
\text{Bel}(x_t) = \eta P(z_t \mid x_t) \text{Bel}(x_t)
\]

Note: Bayes filters also work on continuous state spaces (replace sum by integral)

Example: Localization

- Discrete state \( x \in \{1, 2, \ldots, w\} \times \{1, 2, \ldots, h\} \)
- Belief distribution can be represented as a grid
- This is also called a histogram filter

\[
P(x_{ij}) = a_{ij}
\]

Example: Localization

- Action \( u \in \{\text{north, east, south, west}\} \)
- Robot can move one cell in each time step
- Actions are not perfectly executed
- Example: move east
\[
x_{t-1} = \begin{bmatrix} \cdot \end{bmatrix}, u = \text{east} \Rightarrow \begin{bmatrix} \cdot \end{bmatrix}
\]

60% success rate, 10% to stay/move too far/move one up/move one down
Example: Localization

- Let’s start a simulation run... (shades are hand-drawn, not exact!)

Example: Localization

- t=0
- Prior distribution (initial belief)
- Assume we know the initial location (if not, we could initialize with a uniform prior)

Example: Localization

- t=1, u=east, z=no-marker
- Bayes filter step 1: Apply motion model

Example: Localization

- t=2, u=east, z=marker
- Bayes filter step 2: Apply observation model

Example: Localization

- t=2, u=east, z=marker
- Bayes filter step 2: Apply motion model
- Question: Where is the robot?
Bayes Filter - Summary

- Markov assumption allows efficient recursive Bayesian updates of the belief distribution
- Useful tool for estimating the state of a dynamic system
- Bayes filter is the basis of many other filters
  - Kalman filter
  - Particle filter
  - Hidden Markov models
  - Dynamic Bayesian networks
  - Partially observable Markov decision processes (POMDPs)

Kalman Filter

- Bayes filter with continuous states
- State represented with a normal distribution
- Developed in the late 1950’s
- Kalman filter is very efficient (only requires a few matrix operations per time step)
- Applications range from economics, weather forecasting, satellite navigation to robotics and many more
- Most relevant Bayes filter variant in practice → exercise sheet 2

Normal Distribution

- Univariate normal distribution
  \[ X \sim \mathcal{N}(\mu, \sigma) \]
  \[ p(X = x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \]

- Multivariate normal distribution
  \[ X \sim \mathcal{N}(\mu, \Sigma) \]
  \[ p(x) = \mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right) \]

- Example: 2-dimensional normal distribution

Properties of Normal Distributions

- Linear transformation → remains Gaussian
  \[ X \sim \mathcal{N}(\mu, \Sigma), Y \sim AX + B \]
  \[ Y \sim \mathcal{N}(A\mu + B, A\Sigma A^\top) \]

- Intersection of two Gaussians → remains Gaussian
  \[ X_1 \sim \mathcal{N}(\mu_1, \Sigma_1), X_2 \sim \mathcal{N}(\mu_2, \Sigma_2) \]
  \[ p(X_1, X_2) = \mathcal{N}\left(\frac{\Sigma_2}{\Sigma_1 + \Sigma_2} \mu_1 + \frac{\Sigma_1}{\Sigma_1 + \Sigma_2} \mu_2, \frac{1}{\Sigma_1 + \Sigma_2}\right) \]

Linear Process Model

- Consider a time-discrete stochastic process (Markov chain)
Linear Process Model

- Consider a time-discrete stochastic process
- Represent the estimated state (belief) by a Gaussian \( x_t \sim \mathcal{N}(\mu_t, \Sigma_t) \)
- Assume that the system evolves linearly over time, depends linearly on the controls, and has zero-mean, normally distributed process noise with \( \epsilon_t \sim \mathcal{N}(0, \Sigma) \)

\[
x_t = Ax_{t-1} + Bu_t + \epsilon_t
\]

Linear Observations

- Further, assume we make observations that depend linearly on the state and that are perturbed by zero-mean, normally distributed observation noise

\[
z_t = Cx_t + \delta_t
\]

with \( \delta_t \sim \mathcal{N}(0, \Sigma) \)
Kalman Filter

Estimates the state $x_t$ of a discrete-time controlled process that is governed by the linear stochastic difference equation

$$x_t = Ax_{t-1} + Bu_t + \epsilon_t$$

and (linear) measurements of the state

$$z_t = Cx_t + \delta_t$$

with $\delta_t \sim \mathcal{N}(0, R)$ and $\epsilon_t \sim \mathcal{N}(0, Q)$

From Bayes Filter to Kalman Filter

For each time step, do

1. Apply motion model

$$\text{Bel}(x_t) = \int p(x_t \mid x_{t-1}, u_t) \text{Bel}(x_{t-1}) \, dx_{t-1}$$

$$= \mathcal{N}(x_t; \mu_{t-1} + B u_t, A \Sigma A^T + Q)$$

$$= \mathcal{N}(x_t; \bar{\mu}_t, \Sigma_t)$$

Variables and Dimensions

- **State** $x \in \mathbb{R}^n$
- **Controls** $u \in \mathbb{R}^l$
- **Observations** $z \in \mathbb{R}^k$
- **Process equation**

$$x_t = A x_{t-1} + B u_t + \epsilon$$

$$z_t = C x_t + \delta_t$$

- **Measurement equation**

Kalman Filter

- **Initial belief is Gaussian**

$$\text{Bel}(x_0) = \mathcal{N}(x_0; \mu_0, \Sigma_0)$$

- **Next state is also Gaussian (linear transformation)**

$$x_t \sim \mathcal{N}(A x_{t-1} + B u_t, Q)$$

- **Observations are also Gaussian**

$$z_t \sim \mathcal{N}(C x_t, R)$$

From Bayes Filter to Kalman Filter

For each time step, do

1. Apply motion model

$$\text{Bel}(x_t) = \int p(x_t \mid x_{t-1}, u_t) \text{Bel}(x_{t-1}) \, dx_{t-1}$$

$$= \mathcal{N}(x_t; \bar{\mu}_t, \Sigma_t)$$

$$= \mathcal{N}(x_t; \bar{\mu}_t, \Sigma_t)$$

2. Apply sensor model

$$\text{Bel}(x_t) = \frac{\eta p(z_t \mid x_t)}{N(x_t; \bar{\mu}_t, \Sigma_t)} \text{Bel}(x_t)$$

$$= \mathcal{N}(x_t; \bar{\mu}_t + K_t(z_t - C \bar{\mu}), (I - K_t C) \Sigma)$$

$$= \mathcal{N}(x_t; \bar{\mu}_t, \Sigma_t)$$

with $K_t = \Sigma_t C^T (C \Sigma_t C^T + R)^{-1}$
Kalman Filter

For each time step, do

1. Apply motion model
   \[
   \begin{align*}
   \bar{\mu}_t &= A\bar{\mu}_{t-1} + Bu_t \\
   \bar{\Sigma}_t &= A\bar{\Sigma}_{t-1}A^T + Q
   \end{align*}
   \]

2. Apply sensor model
   \[
   \begin{align*}
   \mu_t &= \bar{\mu}_t + K_t(z_t - C\bar{\mu}_t) \\
   \Sigma_t &= (I - K_tC)\bar{\Sigma}_t \\
   \text{with } K_t &= \bar{\Sigma}_tC^T(C\bar{\Sigma}_tC^T + R)^{-1}
   \end{align*}
   \]

Nonlinear Dynamical Systems

- Most realistic robotic problems involve nonlinear functions
- Motion function
  \[x_t = g(u_t, x_{t-1})\]
- Observation function
  \[z_t = h(x_t)\]

Extended Kalman Filter

For each time step, do

1. Apply motion model
   \[
   \begin{align*}
   \bar{\mu}_t &= g(\mu_{t-1}, u_t) \\
   \bar{\Sigma}_t &= G_t\bar{\Sigma}_{t-1}G_t^T + Q \text{ with } G_t &= \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}}
   \end{align*}
   \]

2. Apply sensor model
   \[
   \begin{align*}
   \mu_t &= \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t)) \\
   \Sigma_t &= (I - K_tC)\bar{\Sigma}_t \\
   \text{with } K_t &= \bar{\Sigma}_tH_t^T(H_t\bar{\Sigma}_tH_t^T + R)^{-1} \text{ and } H_t = \frac{\partial h(\bar{\mu}_t)}{\partial x_t}
   \end{align*}
   \]

Taylor Expansion

- Solution: Linearize both functions
- Motion function
  \[g(x_{t-1}, u_t) \approx g(\mu_{t-1}, u_t) + \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}}(x_{t-1} - \mu_{t-1}) = g(\mu_{t-1}, u_t) + G_t(x_{t-1} - \mu_{t-1})\]
- Observation function
  \[h(x_t) \approx h(\bar{\mu}_t) + \frac{\partial h(\bar{\mu}_t)}{\partial x_t}(x_t - \mu_t) = h(\bar{\mu}_t) + H_t(x_t - \mu_t)\]

Example

- 2D case
- State \(\mathbf{x} = (x \ y \ \psi)^T\)
- Odometry \(\mathbf{u} = (\dot{x} \ \dot{y} \ \dot{\psi})^T\)
- Observations of visual marker \(\mathbf{z} = (x \ y \ \psi)^T\) (relative to robot pose)
- Fixed time intervals \(\Delta t\)
Example

- Motion function
  \[ g(x, u) = \begin{pmatrix} x + (\cos(\psi)\dot{x} - \sin(\psi)\dot{y})\Delta t \\ y + (\sin(\psi)\dot{x} + \cos(\psi)\dot{y})\Delta t \\ \psi + \dot{\psi}\Delta t \end{pmatrix} \]
- Derivative of motion function
  \[ G = \frac{\partial g(x, u)}{\partial x} = \begin{pmatrix} 1 & 0 & (-\sin(\psi)\dot{x} - \cos(\psi)\dot{y})\Delta t \\ 0 & 1 & (\cos(\psi)\dot{x} + \sin(\psi)\dot{y})\Delta t \end{pmatrix} \]

Example

- Observation Function (→ Sheet 2)
  \[ h(x) = \ldots \]
  \[ H = \frac{\partial h(x)}{\partial x} = \ldots \]

Example

- Dead reckoning (no observations)
- Large process noise in \( x+y \)

Example

- Dead reckoning (no observations)
- Large process noise in \( x+y+yaw \)

Example

- Now with observations (limited visibility)
- Assume robot knows correct starting pose

Example

- What if the initial pose (\( x+y \)) is wrong?
Example

- What if the initial pose \((x+y+yaw)\) is wrong?

Lessons Learned Today

- Observations and actions are inherently noisy
- Knowledge about state is inherently uncertain
- Probability theory
- Probabilistic sensor and motion models
- Bayes Filter, Histogram Filter, Kalman Filter, Examples

Control Architecture

Visual Navigation for Flying Robots

Robot Control

Dr. Jürgen Sturm
**Agenda for Today**

- Motors
- Motor Controllers
- Kinematics and Dynamics
- Linear Control

**DC Motors**

- Maybe you have built one in school
- Stationary permanent magnet
- Electromagnet induces torque
- Split ring switches direction of current

**Brushless Motors**

- Used in most quadrocopters
- Permanent magnets on the axis
- Electromagnets on the outside
  → Does not require brushes (less maintenance)

**Motor Controller**

- Micro controller estimates rotation and generates PWM signal
- AC signal generator (inverter) generates motor phases

**Pulse Width Modulation (PWM)**

- Protocol used to control motor speed
- Remote controls typically output PWM
I2C Protocol

- Serial data line (SDA) + serial clock line (SCL)
- All devices connected in parallel
- 7-10 bit address, 100-3400 kbit/s speed
- Used by Mikrocopter for motor control

Attitude + Motor Controller Boards

- Example: Mikrokopter Platform

- Example: Ardrone Platform

Attitude + Motor Controller Boards

- Example: Ardrone Platform

Control Architecture

- Kinematics and Dynamics

- Example: 1D Kinematics
Dynamics - Essential Equations

- Force (Kraft)
  \[ m \ddot{x} = \sum F_i \]
- Torque (Drehmoment)
  \[ J \alpha = \sum \tau_i \]

forces

\[ \begin{align*}
  F_x &= m \ddot{x} \\
  F_y &= F_{\text{grav}} = mg \\
  F_z &= F_{\text{drag}} = c_W A \frac{1}{2} \rho \dot{x} \\
  F_{\text{spring}} &= K (x - x_{\text{eq}}) \\
  F_{\text{stiction}} &= c_s \text{sign} \dot{x} \\
  F_{\text{damping}} &= D \dot{x} \\
  F_{\text{friction}} &= c_f \dot{x} \\
  F_{\text{magnetic}} &= K \end{align*} \]

example: spring-damper system

- Combination of spring and damper
- Forces \[ F = F_{\text{damping}} + F_{\text{spring}} \]
- Resulting dynamics \[ m \ddot{x} = D \dot{x} + K (x - x_{\text{eq}}) \]

\[ \begin{align*}
  \tau &= F \times r \\
  \tau_{\text{net}} &= \sum \tau_i \\
  \tau = J \alpha \\
  (with \ \alpha = \frac{d\theta}{dt}, \ J \ \text{moment of inertia}) \\
  \text{Friction same as before...} \\
\]

Dynamics of a Quadrocopter

- Each propeller induces force and torque by accelerating air
- Gravity pulls quadrocopter downwards

\[ \begin{align*}
  F_{\text{thrust}} &= F_1 + F_2 + F_3 + F_4 \\
  F_{\text{grav}} &= \text{downwards} \\
\]
### Vertical and Horizontal Acceleration

- **Thrust**
  \[ F_{\text{thrust}} = F_1 + F_2 + F_3 + F_4 \]

- **Acceleration**
  \[ \ddot{x}_{\text{global}} = \frac{(R_{\text{RPY}} F_{\text{thrust}} - F_{\text{grav}})}{m} \]

### Pitch (and Roll)

- Attitude changes when opposite motors generate unequal thrust
- Induced torque \( \tau = (F_1 - F_3) \times r \)
- Induced angular acceleration \( \alpha = J^{-1} \tau \)

### Yaw

- Each propeller induces torque due to rotation and the interaction with the air
- Induced torque \( \tau = \tau_1 - \tau_2 + \tau_3 - \tau_4 \)
- Induced angular acceleration

### Assumptions of Cascaded Control

- Dynamics of inner loops is so fast that it is not visible from outer loops
- Dynamics of outer loops is so slow that it appears as static to the inner loops
Cascaded Control Example

- Motor control happens on motor boards (controls every motor tick)
- Attitude control implemented on microcontroller with hard real-time (at 1000 Hz)
- Position control (at 10 – 250 Hz)
- Trajectory (waypoint) control (at 0.1 – 1 Hz)

Feedback Control

- Given:
  - Goal state $x_{des}$
  - Measured state (feedback) $x$
- Wanted:
  - Control signal $u$ to reach goal state
- How to compute the control signal?
Feedback Control - Generic Idea

Controller (Regler)

How can we correct?

Turn hotter (not colder)

Plant (Regelstrecke)

Desired value 35°

Sensor

Measured temperature

How hot is it?

Feedback Control - Example

\[ x_{des} \rightarrow Controller \quad u_t = K(x_{des} - z) \rightarrow Plant \quad x_t = x_{t-1} + u_t + \epsilon_t \]

Measurement

Proper Control with Measurement Noise

\[ \text{Lower the gain… (K=0.15)} \]

What happens if sign is messed up?

\[ \text{Check K=-0.5} \]

Measurement Noise

\[ \text{What effect has noise in the measurements?} \]

\[ \text{Poor performance for K=1} \]

\[ \text{How can we fix this?} \]

What do High Gains do?

\[ \text{High gains are always problematic (K=2.15)} \]

What do High Gains do?

How hot is it?

Sensor

Measured temperature

How hot is it?

Feedback Control - Example

Controller

Plant

Measurement

Proper Control with Measurement Noise

\[ \text{Lower the gain… (K=0.15)} \]

What happens if sign is messed up?

\[ \text{Check K=-0.5} \]
**Saturation**
- In practice, often the set of admissible controls $u$ is bounded
- This is called (control) saturation

**Block Diagram**

**Delays**
- In practice most systems have delays
- Can lead to overshoots/oscillations/destabilization
- One solution: lower gains (why is this bad?)

**Delays**
- What is the total dead time of this system?

**Delays**
- What is the total dead time of this system?
- Can we distinguish delays in the measurement from delays in actuation?

**Delays**
- Can we distinguish delays in the measurement from delays in actuation? No!
Smith Predictor

- Allows for higher gains
- Requires (accurate) model of plant

Smith Predictor

- Plant model is available
- 5 seconds delay
- Results in perfect compensation
- Why is this unrealistic in practice?

Smith Predictor

- Time delay (and plant model) is often not known accurately (or changes over time)
- What happens if time delay is overestimated?

Smith Predictor

- Time delay (and plant model) is often not known accurately (or changes over time)
- What happens if time delay is underestimated?

Position Control

- Consider a rigid body
- Free floating in 1D space, no gravity
- How does this system evolve over time?
Rigid Body Kinematics

- Consider a rigid body
- Free floating in 1D space, no gravity
- In each time instant, we can apply a force $F$
- Results in acceleration $\ddot{x} = F/m$
- Desired position $x_{\text{des}} = 1$

P Control

- What happens for this control law?
  $$u_t = K(x_{\text{des}} - x_{t-1})$$
- This is called proportional control

PD Control

- What happens for this control law?
  $$u_t = K_P(x_{\text{des}} - x_{t-1}) + K_D(\ddot{x}_{\text{des}} - \ddot{x}_{t-1})$$
- Proportional-Derivative control
**PD Control**

- What happens for this control law?
  \[ u_t = K_P(x_{des} - x_t) + K_D(\dot{x}_{des} - \dot{x}_t) \]

- What if we set **higher** gains?

- What if we set **lower** gains?

**Gravity compensation**

- Add as an additional term in the control law
  \[ u_t = K_P(x_{des} - x_t) + K_D(\dot{x}_{des} - \dot{x}_t) + F_{grav} \]

- Any known (inverse) dynamics can be included

**PD Control**

- What happens when we add gravity?

**PID Control**

- Idea: Estimate the system error (bias) by integrating the error
  \[ u_t = K_P(x_{des} - x_t) + K_D(\dot{x}_{des} - \dot{x}_t) + K_I \int x_{des} - x_t \, dt \]

- Proportional+Derivative+Integral Control
**PID Control**

- Idea: Estimate the system error (bias) by integrating the error
  \[ u_t = K_P(x_{des} - x_t) + K_D(\dot{x}_{des} - \dot{x}_t) + K_I \int_0^t x_{des} - x_t \, dt \]
- Proportional+Derivative+Integral Control
- For steady state systems, this can be reasonable
- Otherwise, it may create havoc or even disaster (wind-up effect)

**Example: Wind-up effect**

- Quadrocopter gets stuck in a tree → does not reach steady state
- What is the effect on the I-term?

**De-coupled Control**

- So far, we considered only single-input, single-output systems (SISO)
- Real systems have multiple inputs + outputs
- MIMO (multiple-input, multiple-output)
- In practice, control is often de-coupled

**How to Choose the Coefficients?**

- Gains too large: overshooting, oscillations
- Gains too small: long time to converge
- Heuristic methods exist
- In practice, often tuned manually

**Example: Ardrone**

Cascaded control
- Inner loop runs on embedded PC and stabilizes flight
- Outer loop runs externally and implements position control

**Ardrone: Inner Control Loop**

- Plant input: motor torques
  \[ u_{inner} = (\tau_1, \tau_2, \tau_3, \tau_4) \]
- Plant output: roll, pitch, yaw rate, z velocity
  \[ x_{inner} = (\omega_x, \omega_y, \omega_z, z) \]

- Attitude (measured using gyro + accelerometer)
- Altitude (measured using ultrasonic distance sensor + attitude)
**Ardrone: Inner Control Loop**

- **Plant input:** motor torques
  \[
  \mathbf{u}_{\text{inner}} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 \end{bmatrix}^T
  \]
- **Plant output:** roll, pitch, yaw rate, z velocity
  \[
  \mathbf{x}_{\text{inner}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \psi \end{bmatrix}^T
  \]

**Ardrone: Outer Control Loop**

- **Outer loop sees inner loop as a plant (black box)**
- **Plant input:** roll, pitch, yaw rate, z velocity
  \[
  \mathbf{u}_{\text{outer}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T
  \]
- **Plant output:**
  \[
  \mathbf{x}_{\text{outer}} = \begin{bmatrix} x & y & z & \psi \end{bmatrix}^T
  \]

**Mechanical Equivalent**

- PD Control is equivalent to adding spring-dampers between the desired values and the current position

**PID Control – Summary**

PID is the most used control technique in practice

- **P control** → simple proportional control, often enough
- **PI control** → can compensate for bias (e.g., wind)
- **PD control** → can be used to reduce overshoot (e.g., when acceleration is controlled)
- **PID control** → all of the above

**Advanced Control Techniques**

What other control techniques do exist?

- Adaptive control
- Robust control
- Optimal control
- Linear-quadratic regulator (LQR)
- Reinforcement learning
- Inverse reinforcement learning
- ... and many more
Optimal Control

- Find the controller that provides the best performance
- Need to define a measure of performance
- What would be a good performance measure?
  - Minimize the error?
  - Minimize the controls?
  - Combination of both?

Linear Quadratic Regulator (LQR)

- Given:
  - Discrete-time linear system
    \[ x_{k+1} = Ax_k + Bu_k \]
  - Quadratic cost function
    \[ J = \sum_{k=0}^{\infty} (x_k^TQx_k + u_k^TRu_k) \]

Goal: Find the controller with the lowest cost → LQR control

Reinforcement Learning

- Note that in principle, any cost function can be used
- Sometimes, it is easier to specify a reward function \( r(x_t, u_t) \)
- Example:
  \[ r(x_t, u_t) = \begin{cases} 1 & \text{if } x_{\text{goal}} = x_t \\ 0 & \text{otherwise} \end{cases} \]

- A policy (=controller) defines which action to take in a particular state
  \[ \pi(x) = u \]
Policy Evaluation

- The optimal policy maximizes the expected future reward
- How can we estimate the expected future reward?
- How can we find the optimal policy?
- Closer look at two methods
  - Q learning
  - Policy gradient methods

Reinforcement Learning

- Q learning
  - Learn the value function for each state-action pair
  - Needs compact representation of value function (e.g., neural networks)
  - Examples: TD-Gammon (mid-90’s), Inverted pendulum, ...

Policy gradient methods

- Policy is parameterized
- Analytic gradient typically not available
- Simulation-based optimization

Inverse Reinforcement Learning

- Parameterized reward function
- Learn these parameters from expert demonstrations and refine
- Example: [Abbeel and Ng, ICML 2010]

Lessons Learned Today

- Brushless Motors
- Motor Controllers
- Cascaded Control
- PID Control
- Advanced Control Techniques
Motivation

Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots

Visual Motion Estimation

Dr. Jürgen Sturm

Quick geometry recap
Image filters
2D image alignment
Corner detectors
Kanade-Lucas-Tomasi tracker
2D motion estimation
Interesting research papers from ICRA and ROSCon

Recap: Perspective Projection

3D to 2D Perspective Projection

3D point \( p \) (in the camera frame)
2D point \( x \) (on the image plane)
Pin-hole camera model

\[
\tilde{x} = \lambda \tilde{r} = p
\]

Remember, \( \tilde{x} \) is homogeneous, need to normalize

\[
\tilde{x} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} \Rightarrow x = \begin{pmatrix} \tilde{x}/\tilde{z} \\ \tilde{y}/\tilde{z} \end{pmatrix}
\]
Camera Intrinsics

- So far, 2D point is given in meters on image plane (located in 1m distance from origin)
- But: we want 2D point be measured in pixels (as the sensor does)

\[ \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \]

- Focal length \( f_x, f_y \)
- Camera center \( c_x, c_y \)
- Skew \( s \)

Image Plane

- Pixel coordinates \( x \in \Omega \)
- Image plane \( \Omega \subset \mathbb{R}^2 \)

- Example:
  - Discrete case \( x \in [0, W - 1] \times [0, H - 1] \subset \mathbb{N}_0^2 \) (default in this course)
  - Continuous case \( x \in [0, 1] \times [0, 1] \subset \mathbb{R}^2 \)

Image Functions

- We can think of an image as a function \( f : \Omega \mapsto \mathbb{R} \)
- \( f(x) \) gives the intensity at position \( x \)
- Color images are vector-valued functions

\[ f : \Omega \mapsto \mathbb{R}^3 \\
\begin{bmatrix} r(x) \\ g(x) \\ b(x) \end{bmatrix} \]

- Realistically, the image function is only defined on a rectangle and has finite range

\[ f : [0, W - 1] \times [0, H - 1] \mapsto [0, 1] \]

- Image can be represented as a matrix

\[ F_{ij} = f(i, j), f(x, y), f(x) \ldots \]

- Alternative notations

\[ \text{often (row,column)} \]
\[ \text{often (column,row)} \]
Digital Images

- Light intensity is sampled by CCD/CMOS sensor on a regular grid
- Electric charge of each cell is quantized and gamma compressed (for historical reasons)
  \[ V = B^{\frac{1}{\gamma}} \text{ with } \gamma = 2.2 \]
- CRTs / monitors do the inverse \[ B = V^\gamma \]
- Almost all images are gamma compressed → Double brightness results only in a 37% higher intensity value (!)

Aliasing

- High frequencies in the scene and a small fill factor on the chip can lead to (visually) unpleasing effects
- Examples

Rolling Shutter

- Most CMOS sensors have a rolling shutter
- Rows are read out sequentially
- Sensitive to camera and object motion
- Can we correct for this?

Image Filtering

- We want to remove unwanted sources of variation, and keep the information relevant for whatever task we need to solve
  \[ f(i,j) \rightarrow g(i,j) \]
- Example tasks: de-noising, (de-)blurring, computing derivatives, edge detection, ...

Linear Filtering

- Each output is a linear combination of all the input values
  \[ g(i,j) = \sum_{k,l} h(i,j,k,l)f(k,l) \]
- In matrix form
  \[ G = H F \]

Spatially Invariant Filtering

- We are often interested in spatially invariant operations
  \[ g(i,j) = f * h = \sum_{k,l} h(i-k,j-l)f(k,l) \]
- Example
Spatially Invariant Filtering

- We are often interested in spatially invariant operations
  \[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

Example

**Important Filters**

- Impulses
- Shifts
- Blurring and de-blurring
  - Gaussian
  - Bilateral filter
  - Motion blur
- Edges
  - Finite difference filter
  - Derivative filter
  - Oriented filters
  - Gabor filter
- ...

**Impulse**

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

**Image shift (translation)**

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

**Image rotation**

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

**Image rotation**

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k, l) \]

Image rotation is a linear operator (why?), but not a spatially invariant operation (why?). There is no convolution.
### Rectangular Filter

\[
g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k, l)
\]

### Gaussian Blur

- Gaussian distribution
  \[
g_g(i, j) = \frac{1}{2\pi\sigma^2} \exp \left( -\frac{i^2 + j^2}{2\sigma^2} \right)
\]
- Example of resulting kernel

### Image Gradient

- The image gradient \( \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^T \) points in the direction of increasing intensity (steepest ascend)
Image Gradient

- The image gradient $\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^T$ points in the direction of increasing intensity (steepest ascend)

\[
\nabla f = \left( \frac{\partial f}{\partial x}, 0 \right)^T \quad \nabla f = \left( 0, \frac{\partial f}{\partial y} \right)^T \quad \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^T
\]

- Gradient direction (related to edge orientation)
- Gradient magnitude (edge strength)

How can we differentiate a digital image $f(x, y)$?

- **Option 1**: Reconstruct a continuous image, then take gradient
- **Option 2**: Take discrete derivative (finite difference filter)
- **Option 3**: Convolve with derived Gaussian (derivative filter)

Finite difference

- First-order central difference (half pixel)
  \[
  \frac{\partial f}{\partial x}(x, y) \approx f(x + 0.5, y) - f(x - 0.5, y)
  \]
  Corresponding convolution kernel: 

- First-order central difference
  \[
  \frac{\partial f}{\partial x}(x, y) \approx \frac{f(x + 1, y) - f(x - 1, y)}{2}
  \]
  Corresponding convolution kernel:

- Differentiate again to get second-order central difference
  \[
  \frac{\partial^2 f}{\partial x^2}(x) \approx f(x + 1) - 2f(x) + f(x - 1)
  \]
  Corresponding convolution kernel:
Example

\[ g(i, j) = f * h = \sum_{k,l} h(i - k, j - l) f(k,l) \]

\[ f(i, j) \ast h(i, j) = g(i, j) \]

(Dense) Motion Estimation

- 2D motion
- 3D motion

Problem Statement

- **Given:** two camera images \( f_0, f_1 \)
- **Goal:** estimate the camera motion \( \mathbf{u} \)

For the moment, let’s assume that the camera only moves in the \( xy \)-plane, i.e., \( \mathbf{u} = (u \ v)^T \)

- Extension to 3D follows

General Idea

1. Define an error metric \( E(\mathbf{u}) \) that defines how well the two images match given a motion vector
2. Find the motion vector with the lowest error
   \[ \mathbf{u}^* = \arg \min_{\mathbf{u}} E(\mathbf{u}) \]

Error Metrics for Image Comparison

- Sum of Squared Differences (SSD)
  \[ E_{\text{SSD}}(\mathbf{u}) = \sum_i (f_1(x_i + \mathbf{u}) - f_0(x_i))^2 = \sum_i e_i^2 \]

with displacement \( \mathbf{u} = (u \ v)^T \)

and residual errors \( e_i = f_1(x_i + \mathbf{u}) - f_0(x_i) \)
Robust Error Metrics

- SSD metric is sensitive to outliers
- Solution: apply a (more) robust error metric

$$E_{\text{SRD}}(u) = \sum_i \rho(f_1(x_i + u) - f_0(x_i)) = \sum_i \rho(e_i)$$

Robust Error Metrics

- Sum of absolute differences (SAD, L1 norm)
  $$\rho_{\text{SAD}}(e) = |e|$$

- Sum of truncated errors
  $$\rho_{\text{trunc}}(e) = \begin{cases} e^2 & \text{if } |e| < b \\ b^2 & \text{otherwise} \end{cases}$$

- Geman-McClure
  $$\rho_{\text{gm}}(e) = \frac{x^2}{1 + x^2}$$

Windowed SSD

- Images (and image patches) have finite size
- Standard SSD has a bias towards smaller overlaps (less error terms)
- Solution: divide by the overlap area
- Root mean square error
  $$E_{\text{RMS}}(u) = \sqrt{E_{\text{SSD}}/A}$$

Exposure Differences

- Images might be taken with different exposure (auto shutter, white balance, ...)
- Bias and gain model
  $$f_1(x + u) = (1 + \alpha)f_0(x) + \beta$$
- With SSD we get
  $$E_{\text{BG}}(u) = \sum_i (f_1(x_i + u) - (1 + \alpha)f_0(x_i) + \beta)^2$$
  $$= \sum_i \alpha f_0(x) + \beta - e_i^2$$

Cross-Correlation

- Maximize the product (instead of minimizing the differences)
  $$E_{\text{CC}}(u) = -\sum_i f_0(x_i)f_1(x_i + u)$$

- Normalized cross-correlation (between -1..1)
  $$E_{\text{NCC}}(u) =$$
  $$\frac{-\sum_i (f_0(x_i) - \text{mean} f_0)(f_1(x_i + u) - \text{mean} f_1)}{\sqrt{\text{var} f_0 \text{var} f_1}}$$
General Idea

1. Define an error metric $E(u)$ that defines how well the two images match given a motion vector.

2. Find the motion vector with the lowest error

$$u^* = \arg \min_u E(u)$$

Finding the minimum

- Full search (e.g., ±16 pixels)
- Gradient descent
- Hierarchical motion estimation

Hierarchical motion estimation

- Construct image pyramid
- Estimate motion on coarse level
- Use as initialization for next finer level

Motion Estimation

- Perform Gauss-Newton minimization on the SSD energy function (Lucas and Kanade, 1981)
- Gauss-Newton minimization
  - Linearize residuals w.r.t. to camera motion
  - Yields quadratic cost function
  - Build normal equations and solve linear system

Motion Estimation

- Taylor expansion of energy function

$$E_{\text{Lk-SSD}}(u + \Delta u) = \sum_i (f_1(x_i + u + \Delta u) - f_0(x_i))^2$$

$$\approx \sum_i (f_1(x_i + u) + J_1(x + u)\Delta u - f_0(x_i))^2$$

$$= \sum_i (J_1(x + u)\Delta u + e_i )^2$$

with $J_1(x + u) = \nabla f_1(x + u) = \left( \frac{\partial f_1}{\partial x}, \frac{\partial f_1}{\partial y} \right)(x_i + u)$

Least Squares Minimization

- Goal: Minimize

$$E(u + \Delta u) = \sum_i (J_1(x_i + u)\Delta u + e_i )^2$$

- Solution: Compute derivative and set to zero

$$\frac{\partial E(u + \Delta u)}{\partial \Delta u} = 2A\Delta u + 2b = 0$$

with $A = \sum_i J_1^T(x_i + u)J_1(x + u)$

and $b = \sum_i e_i J_1^T(x_i + u)$
Least Squares Minimization

1. Compute $A, b$ from image gradients using

$$A = \left( \sum f_x^2 \sum f_y^2 \right) \quad b = \left( \sum f_x f_y \right)$$

with $f_x = \frac{\partial f_1(x)}{\partial x}$, $f_y = \frac{\partial f_1(x)}{\partial y}$

and $f_t = \frac{\partial f_1(x)}{\partial t} \left[ \approx f_1(x) - f_0(x) \right]$

2. Solve $A \Delta u = -b$

$$\Rightarrow \Delta u = -A^{-1}b$$

Covariance of the Estimated Motion

- Assuming (small) Gaussian noise in the images

$$f_{obs}(x_i) = f_{true}(x_i) + \epsilon_i$$

with $\epsilon_i \sim N(0, \sigma^2)$

- ... results in uncertainty in the motion estimate with covariance (e.g., useful for Kalman filter)

$$\Sigma_u = \sigma^2 A^{-1}$$

Optical Computer Mouse (since 1999)

- E.g., ADNS3080 from Agilent Technologies, 2005
  - 6400 fps
  - 30x30 pixels
  - 4 USD

Image Patches

- Sometimes we are interested of the motion of a small image patches
- **Problem**: some patches are easier to track than others
- What patches are easy/difficult to track?
- How can we recognize “good” patches?

Example

- Let’s look at the shape of the energy
**Corner Detection**

\[ A = \left( \begin{array}{cc} \sum f_x^2 & \sum f_x f_y \\ \sum f_x f_y & \sum f_y^2 \end{array} \right) \]

- **Idea:** Inspect eigenvalues \( \lambda_1, \lambda_2 \) of Hessian \( A \)
  - \( \lambda_1, \lambda_2 \) small \( \rightarrow \) no point of interest
  - \( \lambda_1 \) large, \( \lambda_2 \) small \( \rightarrow \) edge
  - \( \lambda_1, \lambda_2 \) large \( \rightarrow \) corner

- Harris detector (does not need eigenvalues)
  \[ \lambda_1 \lambda_2 > \kappa (\lambda_1 + \lambda_2)^2 \Leftrightarrow \det(A) > \kappa \text{trace}^2(A) \]

- Shi-Tomasi (or Kanade-Lucas)
  \[ \min(\lambda_1, \lambda_2) > \kappa \]

**Other Detectors**

- Förstner detector (localize corner with sub-pixel accuracy)
- FAST corners (learn decision tree, minimize number of tests \( \rightarrow \) super fast)
- Difference of Gaussians / DoG (scale-invariant detector)
- ...

**Kanade-Lucas-Tomasi (KLT) Tracker**

- **Algorithm**
  1. Find (Shi-Tomasi) corners in first frame and initialize tracks
  2. Track from frame to frame
  3. Delete track if error exceeds threshold
  4. Initialize additional tracks when necessary
  5. Repeat step 2-4

- KLT tracker is highly efficient (real-time on CPU) but provides only sparse motion vectors
- Dense optical flow methods require GPU

**Interesting Papers from ICRA 2013**

Real-time Motion Tracking on a Cellphone using Inertial Sensing and a Rolling-Shutter Camera

Mengyang Li, Hung Hyoung Kim and Anantrasit D. Mamatko

Dept. of Electrical Engineering, University of California, Riverside

E-mail: mengyang.acucla, hhkim.acsucla, mamatko.acucla

**Abstract**— All existing methods for vision-aided inertial navigation assume a device with a global sensor, in which all the inertial sensors (IMU) are placed in a single enclosure. In this paper we describe the first, to the best of our knowledge, method for vision-aided inertial navigation relatively using the onboard IMU. To this end, we introduce a complementary approach for handling the rollotherapy effect, avoiding any original intro-

![Interesting Papers from ICRA 2013](image.png)
Smartphone as a Sensor Platform

- Quadcore
- Multiple cameras
- Accelerometer
- Gyroscopes
- GPS
- Wireless
- Relatively cheap

Challenges

- Rolling shutter camera
- Poor synchronization between camera and IMU

Kalman Filter

- Kalman state contains
  - IMU pose
  - last m camera poses
  - We need to define
    - Motion model
    - Observation model

Kalman Filter: Motion Model

- Motion model (IMU)
  - Angular velocity
  - Linear acceleration

Kalman Filter: Observation Model

- Observations from KLT tracker
  - 2D observations of 3D points (3D positions estimated by triangulation, covered next week)
  - Observation function
  - Observation function with rolling shutter comp.
Results

[Li et al., ICRA ’13]

- Galaxy S2 (live demo at ICRA with S3)
- IMU updates at 90 Hz (downsampled)
  - Gyroscope at 106 Hz
  - Accelerometer at 93 Hz
- Images at 15 Hz
  - Read-out time 32msec
- 610m and 900m trajectory
- Drift less than 0.8%, absolute scale from IMU

Dr. Jürgen Sturm, Computer Vision Group, TUM

Commercial Solutions

- Pix4flow from ETH/3D robotics
  1 camera, IMU, ultrasound, 150 EUR
- Parrot Mainboard + Navigation board
  1 camera, IMU, ultrasound, 210 USD

Talks at ROSCon 2013

- ROS Conference (in conjunction with ICRA)
- Talk by Chad Rockey (Willow Garage) on
  Android sensors driver
- Android app (Google Play store or github)
  Provides:
  - `/android/imu`
  - `/android/fix` (GPS)
  - `/camera/camera_info`
  - `/camera/image/compressed`

Dr. Jürgen Sturm, Computer Vision Group, TUM

Android Sensors Driver

This Android app aims to provide an easy way to use Android-enabled devices in ROS. The app currently supports publishing of GPS fixes as `sensor_msgs/GPSFix` and accelerometer/magnetometer/gyroscope data as

Dr. Jürgen Sturm, Computer Vision Group, TUM
Tilting Propellors
[Ryll et al., ICRA ’13]

Second Experiment:
Sinusoidal rotation around X-axis
Keeping position in place
Frontal Obstacle Avoidance
[Mori and Scherer, ICRA ’13]

Frontal Obstacle Detection and Avoidance With Monocular Vision For Small UAVs
Field Robotics Center
Robotics Institute
Carnegie Mellon University

Learning to Avoid Obstacles
[Ross et al., ICRA ’13]

Learning Monocular Reactive UAV Control in Cluttered Natural Environments

ICRA Proceedings
- All papers of a conference are (usually) collected into so-called proceedings
- Previously: One or more books
- Today: USB sticks or online
- Will put ICRA proceedings online (see website)
- Remember password (or ask by email)

Ideas for Your Mini-Project
- Person following (colored shirt or wearing a marker)
- Flying camera for taking group pictures (possibly using the OpenCV face detector)
- Fly through a hula hoop (brightly colored, white background)
- Navigate through a door (brightly colored)
- Navigate from one room to another (using ground markers)
- Avoid obstacles using optical flow
- Landing on a marked spot/moving target
- Your own idea here – be creative!
- ...

Lessons Learned Today
- How to estimate the translational motion from camera images
- Which image patches are easier to track than others
- How to estimate 2D motion from camera images
- Summary of ICRA and ROSCon papers/talks
VISNAV Oral Team Exam

Date and Time | Student Name | Student Name | Student Name
--- | --- | --- | ---
Mon, July 29, 10am |  |  | 
Mon, July 29, 11am |  |  | 
Mon, July 29, 2pm |  |  | 
Mon, July 29, 3pm |  |  | 
Mon, July 29, 4pm |  |  | 
Tue, July 30, 10am |  |  | 
Tue, July 30, 11am |  |  | 
Tue, July 30, 2pm |  |  | 
Tue, July 30, 3pm |  |  | 
Tue, July 30, 4pm |  |  | 

Will put up this list in front of our secretary's office (02.09.052)

ICRA Papers+Videos are Online

Agenda for Today

- **This week:** basic ingredients of a visual SLAM system
  - Feature detection, descriptors and matching
  - Place recognition
  - 3D motion estimation
- **Next week:** bundle adjustment, graph SLAM, stereo cameras, Kinect
- **In two weeks:** map representations, mapping and (dense) 3D reconstruction

Kanade-Lucas-Tomasi (KLT) Tracker

- **Algorithm**
  1. Find (Shi-Tomasi) corners in first frame and initialize tracks
  2. Track from frame to frame
  3. Delete track if error exceeds threshold
  4. Initialize additional tracks when necessary
  5. Repeat step 2-4
- KLT tracker is highly efficient (real-time on CPU) but provides only sparse motion vectors
- Can use coarse-to-fine for larger motions

Last week: KLT Tracker

Kanade-Lucas-Tomasi (KLT) Tracker

- **Algorithm**
  1. Find (Shi-Tomasi) corners in first frame and initialize tracks
  2. Track from frame to frame
  3. Delete track if error exceeds threshold
  4. Initialize additional tracks when necessary
  5. Repeat step 2-4
- KLT tracker is highly efficient (real-time on CPU) but provides only sparse motion vectors
- Can use coarse-to-fine for larger motions

Agenda for Today

- **This week:** basic ingredients of a visual SLAM system
  - Feature detection, descriptors and matching
  - Place recognition
  - 3D motion estimation
- **Next week:** bundle adjustment, graph SLAM, stereo cameras, Kinect
- **In two weeks:** map representations, mapping and (dense) 3D reconstruction
Visual Odometry

[Li et al., ICRA ‘13]

Limitations

- Tracking is based on image gradients (dx/dy/dt)
  - Only works for small motions
  - Preferably high frame rate
  - Cannot recover when tracks are lost

- How can we recognize previously seen patches?

Example: How to Build a Panorama Map

- We need to match (align) images
- Global methods sensitive to occlusion, lighting, parallax effects
- How would you do it by eye?

Matching with Features

- Detect features in both images
- Find corresponding pairs
- Use these pairs to align images
Matching with Features

- Problem 1:
  We need to detect the same point independently in both images

  ![Image with matching points](image1.png)

  → We need a reliable detector

Matching with Features

- Problem 2:
  For each point correctly recognize the corresponding one

  ![Image with matching points](image2.png)

  → We need a reliable and distinctive descriptor

Ideal Feature Detector

- Always finds the same point on an object, regardless of changes to the image
- Insensitive (invariant) to changes in:
  - Scale
  - Lightning
  - Perspective imaging
  - Partial occlusion

Harris Detector

- Rotation invariance?

  ![Rotation invariance diagram](image3.png)

- Remember from last week
  \[ A = \left( \sum f_x^2 \sum f_y^2 \sum f_x f_y \right) \quad R = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2 \]

- Ellipse rotates but its shape (i.e. eigenvalues) remains the same

  → Corner response R is invariant to rotation
**Harris Detector**

- Invariance to intensity change?

- Partial invariance to additive and multiplicative intensity changes
  - Only derivatives are used → invariance to intensity shift \( I \rightarrow I + b \)
  - Intensity scale \( I \rightarrow aI \):
    Because of fixed intensity threshold on local maxima, only partial invariance

- Not invariant to image scale

**Difference Of Gaussians (DoG)**

- Alternative corner detector that is additionally invariant to scale change

- Approach:
  - Run linear filter (diff. of two Gaussians, \( \sigma_1 = 2\sigma_0 \))
  - Do this at different scales
  - Search for a maximum both in space and scale

**Example: Difference of Gaussians**

\[
\sigma = 1 \\
\sigma = 2 \\
\sigma = 4 \\
\sigma = 8
\]
**SIFT Detector**
- Search for local maximum in space and scale
- Corner detections are invariant to scale change

1. Detect maxima in scale-space
2. Non-maximum suppression
3. Eliminate edge points (check ratio of eigenvalues)
4. For each maximum, fit quadratic function and compute center at sub-pixel accuracy

**Example**
1. Input image 233x189 pixel
2. 832 candidates DoG minima/maxima (visualization indicate scale, orient., location)
3. 536 keypoints remain after thresholding on minimum contrast and principal curvature

**Feature Matching**
- Now, we know how to find **repeatable** corners
- Next question: How can we match them?

**Template Convolution**
- Extract a small as a template
- Convolve image with this template

**Invariances**
- Scaling: No
- Rotation: No (maybe rotate template?)
- Illumination: No (use bias/gain model?)
- Perspective projection: Not really
Scale Invariant Feature Transform (SIFT)

- Lowe, 2004: Transform patches into a canonical form that is invariant to translation, rotation, scale, and other imaging parameters.

Approach
1. Find SIFT corners (position + scale)
2. Find dominant orientation and de-rotate patch
3. Extract SIFT descriptor (histograms over gradient directions)

Select Dominant Orientation
- Create a histogram of local gradient directions computed at selected scale (36 bins)
- Assign canonical orientation at peak of smoothed histogram
- Each key now specifies stable 2D coordinates (x, y, scale, orientation)

SIFT Descriptor
- Compute image gradients over 16x16 window (green), weight with Gaussian kernel (blue)
- Create 4x4 arrays of orientation histograms, each consisting of 8 bins
- In total, SIFT descriptor has 128 dimensions

Feature Matching
Given features in $I_1$, how to find best match in $I_2$?
- Define distance function that compares two features
- Test all the features in $I_2$, find the one with the minimal distance

Feature Distance
How to define the difference between features?
- Simple approach is Euclidean distance (or SSD)
  \[ d(d_1, d_2) = \|d_1 - d_2\| \]
Feature Distance

How to define the difference between features?

- Simple approach is Euclidean distance (or SSD)
  \[ d(d_1, d_2) = \|d_1 - d_2\| \]
- Problem: can give good scores to ambiguous (bad) matches

Efficient Matching

For feature matching, we need to answer a large number of nearest neighbor queries

- Exhaustive search \( O(n^2) \)
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found

Efficient Matching

For feature matching, we need to answer a large number of nearest neighbor queries

- Exhaustive search \( O(n^2) \)
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found
For feature matching, we need to answer a large number of nearest neighbor queries

- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found
- Approximate search
  - Locality sensitive hashing
  - Approximate nearest neighbor
- Vocabularies

Efficient Matching

- Best-bin-first: use priority queue for unchecked leafs

Other Descriptors (for intensity images)

- SIFT (Scale Invariant Feature Transform) [Lowe, 2004]
- SURF (Speeded Up Robust Feature) [Bay et al., 2008]
- BRIEF (Binary robust independent elementary features) [Calonder et al., 2010]
- ORB (Oriented FAST and Rotated Brief) [Rublee et al, 2011]
-...

Efficient Matching

- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Localize query in tree
  - Search nearby leaves until nearest neighbor is guaranteed found

Efficient Matching

- Exhaustive search $O(n^2)$
- Indexing (k-d tree)
  - Approximate search
- Vocabulary trees

Other Descriptors (for intensity images)

- SIFT (Scale Invariant Feature Transform) [Lowe, 2004]
- SURF (Speeded Up Robust Feature) [Bay et al., 2008]
- BRIEF (Binary robust independent elementary features) [Calonder et al., 2010]
- ORB (Oriented FAST and Rotated Brief) [Rublee et al, 2011]
-...
**Example: RGB-D SLAM**  
[Engelhard et al., 2011; Endres et al. 2012]

- Feature descriptor: SURF
- Feature matching: FLANN (approximate nearest neighbor)

**Appearance-based Place Recognition**

- Brute-force matching with all previous images is slow (why?)
- How can we do this faster?

**Object/Scene Recognition**

- Analogy to documents: The content can be inferred from the frequency of visual words

**Bag of Visual Words**

- Visual words = (independent) features
**Bag of Visual Words**
- Visual words = (independent) features
- Construct a dictionary of representative words

**Overview**
- Feature detection and extraction (e.g., SIFT, ...)
- Codewords dictionary
- Image representation (histogram of word occurrences)

**Learning the Dictionary**
- Descriptor vectors (e.g., SIFT, SURF, ...)
- Example patch

**Dictionary of visual words (codebook)**

Each detected feature is assigned to the closest entry in the codebook.

**Learning the Dictionary**
- Clustering, e.g., k-means
- Cluster center = code words
Learning the Visual Vocabulary

Example Image Representation
- Build the histogram by assigning each detected feature to the closest entry in the codebook

Object/Scene Recognition
- Compare histogram of new scene with those of known scenes, e.g., using
  - simple histogram intersection
    \[ \text{score}(p, q) = \sum \min(p_i, q_i) \]
  - naïve Bayes
  - more advanced statistical methods

Example: FAB-MAP
[Cummins and Newman, 2008]

Timing Performance
- Inference: 25 ms for 100k locations
- SURF detection + quantization: 483 ms

Summary: Bag of Words
[Fei-Fei and Perona, 2005; Nister and Stewenius, 2006]
- Compact representation of content
- Highly efficient and scalable
- Requires training of a dictionary
- Insensitive to viewpoint changes/image deformations (inherited from feature descriptor)
Structure From Motion (SfM)

- Now we can retrieve relevant images and compute point correspondences between them.

- What can we use them for?

Four Important SfM Problems

- Camera calibration / resection
  Known 3D points, observe corresponding 2D points, compute camera pose.

- Point triangulation
  Known camera poses, observe 2D point correspondences, compute 3D point.

- Motion estimation
  Observe 2D point correspondences, compute camera pose (up to scale).

- Bundle adjustment / visual SLAM (next week!)
  Observe 2D point correspondences, compute camera pose and 3D points (up to scale).

Four Important SfM Problems

- Each of these problems has many solution algorithms.

- Approaches differ in:
  - Number of minimum points and assumptions of their configuration.
  - Effect of noise (bias).
  - Conditioning.
  - Simplicity vs. accuracy (linear vs. non-linear).

Camera Calibration (Perspective n-Point Problem)

- Given: \( n \) 2D/3D correspondences \( x_i \leftrightarrow p_i \).

- Wanted: \( M = K(R, t) \) such that \( \bar{x}_i = Mp_i \).

- Question: How many DOFs does \( M \) have?

- The algorithm has two parts:
  1. Compute \( M \in \mathbb{R}^{3 \times 4} \).
  2. Decompose \( M \) into \( K, R, t \) via QR decomposition.
Step 1: Estimate M

- \( \hat{x}_i = M p_i \)
- Each correspondence generates two equations:
  \[
  \begin{align*}
  x &= m_{1X}X + m_{1Y}Y + m_{1Z}Z + m_{1W}W \\
  y &= m_{2X}X + m_{2Y}Y + m_{2Z}Z + m_{2W}W \\
  z &= m_{3X}X + m_{3Y}Y + m_{3Z}Z + m_{3W}W \\
  u &= m_{4X}X + m_{4Y}Y + m_{4Z}Z + m_{4W}W
  \end{align*}
  \]
- Multiplying out gives equations linear in the elements of \( M \):
  \[
  \begin{align*}
  (m_{2X}X + m_{2Y}Y + m_{2Z}Z + m_{2W}W)x &= m_{11}X + m_{12}Y + m_{13}Z + m_{14}W \\
  (m_{3X}X + m_{3Y}Y + m_{3Z}Z + m_{3W}W)y &= m_{11}X + m_{12}Y + m_{13}Z + m_{14}W
  \end{align*}
  \]
- Re-arrange in matrix form →

Step 1: Estimate M

- Re-arranged in matrix form:
  \[
  \begin{pmatrix}
  X \ Y \ Z \\
  0 \ 0 \ 0 \\
  X \ Y \ Z \\
  0 \ 0 \ 0
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y \\
  z \\
  u
  \end{pmatrix}
  \begin{pmatrix}
  1 \\
  0 \\
  1 \\
  0
  \end{pmatrix}
  =
  \begin{pmatrix}
  m_{11}X + m_{12}Y + m_{13}Z + m_{14}W \\
  m_{21}X + m_{22}Y + m_{23}Z + m_{24}W \\
  m_{31}X + m_{32}Y + m_{33}Z + m_{34}W
  \end{pmatrix}
  \]
- Concatenate equations for \( n \geq 6 \) correspondences:
  \( A \mathbf{m} = \mathbf{0} \)
- Wanted vector \( \mathbf{m} \) is in the null space of \( A \)
- Initial solution using SVD (vector with least singular value), refine using non-linear min.

Step 2: Recover \( K, R, t \)

- Remember \( M = K(R, t) \)
- The first 3x3 submatrix is the product of an upper triangular and orthogonal (rot.) matrix:
  \[
  K = \begin{pmatrix}
  f_x & s & c_x \\
  0 & f_y & c_y \\
  0 & 0 & 1
  \end{pmatrix}
  \]

Procedure:
1. Factor \( M \) into \( KR \) using QR decomposition
2. Compute translation as \( t = K^{-1} (P_{14}, P_{24}, P_{34})^T \)

Example: ARToolkit Markers (1999)

1. Threshold image
2. Detect edges and fit lines
3. Intersect lines to obtain corners
4. Estimate projection matrix \( M \)
5. Extract camera pose \( R, t \) (assume \( K \) is known)

The final error between measured and projected points is typically less than 0.02 pixels

Triangulation

- Given: \( n \) cameras \( \{ M_j = K_j(R_j, t_j) \} \)
- Point correspondence \( x_{ij}, x_{i1} \)
- Wanted: Corresponding 3D point \( p \)

\[
\begin{align*}
\hat{x}_j &= m_{11}X + m_{12}Y + m_{13}Z + m_{14}W \\
\hat{y}_j &= m_{21}X + m_{22}Y + m_{23}Z + m_{24}W \\
\hat{z}_j &= m_{31}X + m_{32}Y + m_{33}Z + m_{34}W
\end{align*}
\]

- Minimize the residuals:
  \[
  p^* = \arg\min_p \sum_j d(x_j, \hat{x}_j)^2
  \]

Triangulation

- Where do we expect to see \( p = (X \ Y \ Z \ W)^T \)?
Triangulation

- Multiply with denominator gives
- Solve for using:
  - Linear least squares with $W=1$
  - Linear least squares using SVD
  - Non-linear least squares of the residuals (most accurate)

Epipolar Geometry

- Let’s consider two cameras that observe a 3D world point

Epipolar Geometry

- The line connecting both camera centers is called the baseline

Epipolar Geometry

- Given the image of a point in one view, what can we say about its position in another?

Epipolar Geometry

- Left line in left camera frame $p_1 = d_1 \hat{x}_1$
- Right line in right camera frame $p_2 = d_2 \hat{x}_2$
where $\hat{x}_j = K^{-1} \hat{x}_j$ are the (local) ray directions
- Intersection of both lines

This is called the epipolar constraint
Epipolar Geometry

Note: The epipolar constraint holds for every pair of corresponding points \( x_1, x_2 \)

\[ \tilde{x}_2^T E \tilde{x}_1 = 0 \]

where \( E \) is called the essential matrix

\[ E = [t]_x R \in \mathbb{R}^{3 \times 3} \]

3D Motion Estimation

- **Given:** 2 camera images
  - \( n \) point correspondences
- **Wanted:** Camera motion \( R, t \) (up to scale)

- **Solutions:**
  - 8-point algorithm
  - normalized 8-point algorithm
  - 6-point algorithm
  - 5-point algorithm

8-Point Algorithm: General Idea

1. Estimate the essential matrix \( E \) from at least eight point correspondences
2. Recover the relative pose \( R, t \) from \( E \) (up to scale)

Step 1: Estimate \( E \)

- Epipolar constraint \( \tilde{x}_2^T E \tilde{x}_1 = 0 \)
- Written out (with \( x_j = (x_j, y_j, 1)^T \))

\[
\begin{align*}
&x_1x_2e_{11} + y_1x_2e_{12} + x_2e_{13} + \\
&x_1y_2e_{21} + y_1y_2e_{22} + y_2e_{23} + \\
&x_1e_{31} + y_1e_{32} + e_{33} = 0
\end{align*}
\]

- Stack the elements into two vectors

\[
\begin{pmatrix}
  x_1 x_2 & y_1 x_2 & \ldots & 1
\end{pmatrix}^T
\begin{pmatrix}
e_{11} & e_{12} & \ldots & e_{33}
\end{pmatrix}
= 0
\]

Normalized 8-Point Algorithm

[Hartley 1997]

- Noise in the point observations is unequally distributed in the constraints, e.g.,
  - double noise
  - normal noise
  - noise free

- Estimation is sensitive to scaling
- Normalize all points to have zero mean and unit variance
Step 2: Recover $R, t$

- **Note:** The absolute distance between the two cameras can never be recovered from pure images measurements alone!!!
- **Illustration**

  ![Step 2: Recover R, t](image1)

  We can only recover the translation $\hat{t}$ up to scale

Step 2a: Recover $t$

- **Remember:** $E = [t]_x R$
- **Therefore,** $t^T$ is in the null space of $E$

\[ t^T E = t^T [t]_x R = 0 \]

→ Recover $\hat{t}$ (up to scale) using SVD

\[ E = [\hat{t}]_x R = U \Sigma V^T \]

\[ = \begin{pmatrix} u_0 & u_1 & \hat{t} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v_0^T \\ v_1^T \\ v_2^T \end{pmatrix} \]

Step 2b: Recover $R$

Remember, the cross-product $[\hat{t}]_\times$

- ... projects a vector onto a set of orthogonal basis vectors including $\hat{t}$
- ... zeros out the $\hat{t}$ component
- ... rotates the other two by 90°

\[ [\hat{t}]_\times = S Z R_{00p} S^T \]

\[ = \begin{pmatrix} s_0 & s_1 & \hat{t} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_0^T \\ s_1^T \\ \hat{t}^T \end{pmatrix} \]

Step 2b: Recover $R$

- Plug this into the essential matrix equation

\[ E = [t]_x R = S Z R_{00p} S^T R = U \Sigma V^T \]

- **By identifying** $S = U$ and $Z = \Sigma$, we obtain

\[ R_{00p} U^T R = V^T \]

\[ R = U R_{00p}^T V \]

Step 2b: Recover $R$

- Matrices $U, V$ are not guaranteed to be rotations (sign flip still yields a valid SVD)

\[ R = \pm U R_{00p}^T V^T \]

- Identify the correct solution using

  - Select those two solutions with $\det R = 1$
  - Triangulate points in 3D
  - Select the solution with the largest number of points in front of the camera

Summary: 8-Point Algorithm

**Given:** Image pair

**Find:** Camera motion $R, t$ (up to scale)

- Compute correspondences
- Compute essential matrix
- Extract camera motion
Lessons Learned Today

- how to detect and match feature points
- how to efficiently recognize places
- how to estimate the camera pose and to triangulate points

Agenda for Today

- Outlier rejection using RANSAC
- Laser-based motion estimation
- The SLAM problem
- Pose graph SLAM
- Map optimization

Remember: 8-Point Algorithm

**Given:** Image pair

**Find:** Camera motion R,t (up to scale)

- Find at least 8 correspondences
- Compute essential matrix
- Extract camera motion

How To Deal With Outliers?

**Problem:** No matter how good the feature descriptor/matcher is, there is always a chance for bad point correspondences (=outliers)

Robust Estimation

Example: Fit a line to 2D data containing outliers

- Input data is a mixture of
  - Inliers (perturbed by Gaussian noise)
  - Outliers (unknown distribution)
- Let’s fit a line using least squares...
Robust Estimation

Example: Fit a line to 2D data containing outliers

- Input data is a mixture of:
  - Inliers (perturbed by Gaussian noise)
  - Outliers (unknown distribution)
- Least squares fit gives poor results!

RANdom SAmple Consensus (RANSAC) [Fischler and Bolles, 1981]

Goal: Robustly fit a model to a data set \( S \) which contains outliers

Algorithm:
1. Randomly select a (minimal) subset
2. Instantiate the model from it
3. Using this model, classify the all data points as inliers or outliers
4. Repeat 1-3 for \( N \) iterations
5. Select the largest inlier set, and re-estimate the model from all points in this set

Example

- Step 1: Sample a random subset

Example

- Step 2: Fit a model to this subset

Example

- Step 3: Classify points as inliers and outliers (e.g., using a threshold distance)

Example

- Step 4: Repeat steps 1-3 for \( N \) iterations

Iteration 2:

\( \rightarrow 5 \) inliers, 7 outliers
Example

- Step 4: Repeat steps 1-3 for N iterations

Iteration 3: 2 inliers, 10 outliers

Example

- Step 5: Select the best model (most inliers), the re-fit model using all inliers

Best model: Iteration 1 (10 inliers, 2 outliers)

How Many Iterations Do We Need?

- For a probability of success $p$, we need

$$N = \frac{\log(1 - p)}{\log(1 - (1 - \epsilon)^s)}$$

iterations

for subset size $s$ and outlier ratio $\epsilon$

- E.g., for $p=0.99:

<table>
<thead>
<tr>
<th>Required points $s$</th>
<th>Line 2</th>
<th>Plane 3</th>
<th>Essential matrix 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>10 %</td>
<td>20 %</td>
<td>30 %  40 %  50 % 60 % 70 %</td>
</tr>
<tr>
<td>Line</td>
<td>2</td>
<td>3</td>
<td>5     7     11    17   27   49</td>
</tr>
<tr>
<td>Plane</td>
<td>3</td>
<td>4</td>
<td>7     11    19    35   70   169</td>
</tr>
<tr>
<td>Essential matrix</td>
<td>8</td>
<td>9</td>
<td>26    78    272   1177 7025 70188</td>
</tr>
</tbody>
</table>

Summary on RANSAC

- Efficient algorithm to estimate a model from noisy and outlier-contaminated data
- RANSAC is used today very widely
- Often used in feature matching / visual motion estimation
- Many improvements/variants (e.g., PROSAC, MLESAC, ...)

Laser-based Motion Estimation

- So far, we looked at motion estimation (and place recognition) from visual sensors
- Today, we cover motion estimation from range sensors
  - Laser scanner (laser range finder, ultrasound)
  - Depth cameras (time-of-flight, Kinect ...)

Laser Triangulation

Idea:

- Well-defined light pattern (e.g., point or line) projected on scene
- Observed by a line/matrix camera or a position-sensitive device (PSD)
- Simple triangulation to compute distance
Laser Triangulation

- Function principle
  - Depth triangulation: \( z = \frac{L}{d} \)

Example: Neato XV-11

- Specs: 360deg, 10Hz, 30 USD

How Does the Data Look Like?

- Laser Scanner
  - Measures angles and distances to closest obstacles
  - Alternative representation: 2D point set (cloud)
  - Probabilistic sensor model: \( p(z | x) \)

Laser-based Motion Estimation

- How can we best align two laser scans?
  - Exhaustive search
  - Iterative minimization (ICP)
**Exhaustive Search**

- Estimate a map using first scan and sensor model
- Sweep second scan over map, compute correlation and select best pose

---

**Example: Exhaustive Search** [Olson, ICRA '09]

- Multi-resolution correlative scan matching
- Real-time by using GPU
- Remember: SE(2) has 3 DOFs

---

**Does Exhaustive Search Generalize To 3D As Well?**

---

**Iterative Closest Point (ICP)**

- **Given:** Two corresponding point sets (clouds)
  \[ P = \{ p_1, \ldots, p_n \} \]
  \[ Q = \{ q_1, \ldots, q_n \} \]
- **Wanted:** Translation \( t \) and rotation \( R \) that minimize the sum of the squared error
  \[ E(R, t) = \frac{1}{n} \sum_{i=1}^{n} ||p_i - Rq_i - t||^2 \]
  where \( p_i \) and \( q_i \) are corresponding points

---

**Known Correspondences**

**Note:** If the correct correspondences are known, both rotation and translation can be calculated in closed form.

---

**Known Correspondences**

- **Idea:** The center of mass of both point sets has to match
  \[ \bar{p} = \frac{1}{n} \sum_{i} p_i \quad \bar{q} = \frac{1}{n} \sum_{i} q_i \]
- Subtract the corresponding center of mass from every point
- Afterwards, the point sets are zero-centered, i.e., we only need to recover the rotation...
Known Correspondences

- Decompose the matrix
  \[ W = \sum (p_i - \bar{p})(q_i - \bar{q})^T = USV^T \]
  using singular value decomposition (SVD)

- **Theorem**
  If \( \text{rank } W = 3 \), the optimal solution of \( E(R, t) \)
  is unique and given by
  \[ R = UV^T \]
  \[ t = \bar{p} - R\bar{q} \]

  (for proof, see [http://hss.ulb.uni-bonn.de/2006/0912/0912.pdf](http://hss.ulb.uni-bonn.de/2006/0912/0912.pdf), p.34/35)

Unknown Correspondences

- If the correct correspondences are not known, it is generally impossible to determine the optimal transformation in one step

ICP Algorithm

- **Algorithm:** Iterate until convergence
  - Find correspondences
  - Solve for \( R, t \)
  - Converges if starting position is “close enough”

Example: ICP

ICP Variants

Many variants on all stages of ICP have been proposed:
- Selecting and weighting source points
- Finding corresponding points
- Rejecting certain (outlier) correspondences
- Choosing an error metric
- Minimization

Performance Criteria

- Various aspects of performance
  - Speed
  - Stability (local minima)
  - Tolerance w.r.t. noise and/or outliers
  - Basin of convergence (maximum initial misalignment)
- Choice depends on data and application
Selecting Source Points

- Use all points
- Random sampling
- Spatially uniform sub-sampling
- Feature-based sampling

Spatially Uniform Sampling

- Density of points usually depends on the distance to the sensor → no uniform distribution
- Can lead to a bias in ICP

Feature-based Sampling

Detect interest points (same as with images)

- Decrease the number of correspondences
- Increase efficiency and accuracy
- Requires pre-processing

Closest Point Matching

- Find closest point in the other point set
- Distance threshold

- Closest-point matching generally stable, but slow

Speeding Up Correspondence Search

Finding closest point is most expensive stage of the ICP algorithm

- Build index for one point set (kd-tree)
- Use simpler algorithm (e.g., projection-based matching)

Projection-based Matching

- Slightly worse performance per iteration
- Each iteration is one to two orders of magnitude faster than closest-point
- Requires point-to-plane error metric
**Error Metrics**

- Point-to-point
- Point-to-plane lets flat regions slide along each other
- Generalized ICP: Assign individual covariance to each data point. [Segal, RSS 2009]

**Minimization**

- Only point-to-point metric has closed form solution(s)
- Other error metrics require non-linear minimization methods

**Example: Real-Time ICP on Range Images**

[Rusinkiewicz and Levoy, 2001]

- Real-time scan alignment
- Range images from structure light system (projector and camera, temporal coding)

**ICP: Summary**

- ICP is a powerful algorithm for calculating the displacement between point clouds
- The overall speed depends most on the choice of matching algorithm
- ICP is (in general) only locally optimal \(\rightarrow\) can get stuck in local minima

**The SLAM Problem**

SLAM is the process by which a robot **builds a map** of the environment and, at the same time, uses the map to **compute its location**:

- Localization: inferring location given a map
- Mapping: inferring a map given a location

The acronym SLAM stands for “simultaneous localization and mapping”.

**SLAM Applications**

SLAM is central to a range of indoor, outdoor, in-air and underwater applications for both unmanned and autonomous vehicles.

**Examples**

- At home: vacuum cleaner, lawn mower
- Air: inspection, transportation, surveillance
- Underwater: reef/environmental monitoring
- Underground: search and rescue
- Space: terrain mapping, navigation
SLAM with Ceiling Camera  
(Samsung Hauzen RE70V, 2008)

Localization, Path planning, Coverage  
(Neato XV11, $300)

SfM vs. SLAM

- Structure from Motion (SfM)
  - Monocular/stereo camera
  - Sometimes uncalibrated sensors (e.g., Flickr images)
- Simultaneous Localization and Mapping (SLAM)
  - Multiple sensors: Laser scanner, ultrasound, monocular/stereo camera, GPS, ...
  - Typically in combination with an odometry sensor
  - Typically pre-calibrated sensors

Remember: 3D Transformations

- Representation as a homogeneous matrix
  \[
  M = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \in \text{SE}(3) \subset \mathbb{R}^{4 \times 4}
  \]
  **Pro:** easy to concatenate and invert
  **Con:** not minimal

- Representation as a twist coordinates
  \[
  \xi = \begin{pmatrix} \omega_x & \omega_y & \omega_z & v_x & v_y & v_z \end{pmatrix}^T \in \mathbb{R}^6
  \]
  **Pro:** minimal
  **Con:** need to convert to matrix for concatenation and inversion

Remember: 3D Rotation as Axis/Angle

- Represent rotation by
  - rotation axis \( \hat{n} \) and
  - rotation angle \( \theta \)
  - 4 parameters \((\hat{n}, \theta)\)
  - 3 parameters \(\omega = \theta \hat{n}\)
    - length is rotation angle
    - also called the angular velocity
    - minimal representation

Remember: 3D Transformations

- From twist coordinates to twist
  \[
  \dot{\xi} = \begin{pmatrix} 0 & -\omega_z & \omega_y & v_z \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_x \\ 0 & 0 & 0 & 0 \end{pmatrix} \in \text{se}(3)
  \]

- Exponential map between \text{se}(3) and \text{SE}(3)
  \[
  M = \exp \dot{\xi} \quad \dot{\xi} = \log M
  \]
  (or compute using Rodriguez’ formula)
Notation

- Camera poses in a minimal representation (e.g., twists)
  \( c_1, c_2, \ldots, c_n \)
- ... as transformation matrices
  \( M_1, M_2, \ldots, M_n \)
- ... as rotation matrices and translation vectors
  \( (R_1, t_1), (R_2, t_2), \ldots, (R_n, t_n) \)

Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame
- **Motion concatenation (for twists)**
  \( c_j = c_i \circ z_{ij} = \log(\exp c_i \exp z_{ij}) \)
- **Motion composition operator (in general)**
  \( c_j = c_i \circ z_{ij} \quad z_{ij} = c_j \circ c_i \)

Loop Closures

- **Idea:** Estimate camera motion from frame to frame
- **Problem:**
  - Estimates are inherently noisy
  - Error accumulates over time \( \rightarrow \) drift
Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame

Graph SLAM

- [Thrun and Montemerlo, 2006; Olson et al., 2006]
  - Use a graph to represent the model
  - Every node in the graph corresponds to a pose of the robot during mapping
  - Every edge between two nodes corresponds to a spatial constraint between them
  - **Graph-based SLAM:** Build the graph and find the robot poses that minimize the error introduced by the constraints

Loop Closures

- **Solution:** Use loop-closures to minimize the drift / minimize the error over all constraints

Graph SLAM Architecture

- Interleaving process of front-end and back-end
- A consistent map helps to determine new constraints by reducing the search space
Problem Definition

- **Given:** Set of relative pose observations $z_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $c_1, \ldots, c_n \in \mathbb{R}^6$  
  $\Rightarrow$ State vector $x = (c_1^T, \ldots, c_n^T)^T \in \mathbb{R}^{6n}$

Map Error

- Observation $z_{ij}$
- Expected relative pose $\bar{z}_{ij} = c_j \oplus c_i$
- Difference between observation and expectation $e_{ij} = z_{ij} \oplus \bar{z}_{ij}$
- Given the correct map $x$, this difference is the result of observation/sensor noise...

Error Function

- **Assumption:** Observation noise is normally distributed
  
  $e_{ij} \sim \mathcal{N}(0, \Sigma_{ij})$

- Error term for one observation (proportional to negative loglikelihood)
  
  $f_{ij}(x) = -\log p(e_{ij}) \propto e_{ij}(x)^T \Sigma_{ij}^{-1} e_{ij}(x)$

- Note: error is a scalar $f_{ij}(x) \in \mathbb{R}$

Non-Linear Optimization Techniques

- Gradient descend
- Gauss-Newton
- Levenberg-Marquardt

Gauss-Newton Method

1. Linearize the error function
2. Compute its derivative
3. Set the derivative to zero
4. Solve the linear system
5. Iterate this procedure until convergence
Linearization and Derivation

- Error function
  \[ f(x) = \sum_{ij} f_{ij}(x) = \sum_{ij} e_{ij}(x) \Sigma_{ij}^{-1} e_{ij}(x) \]
- Linearize the error function around the initial guess
  \[ f(x + \Delta x) = \sum_{ij} e_{ij}(x + \Delta x) \Sigma_{ij}^{-1} e_{ij}(x + \Delta x) \]

Let’s look at this term first...

Linearizing the Error Function

- Approximate the error function around an initial guess \( x \in \mathbb{R}^n \) using Taylor expansion
  \[ e_{ij}(x + \Delta x) \approx e_{ij}(x) + J_{ij} \Delta x \quad (\in \mathbb{R}^6) \]
  with increment
  \[ \Delta x \in \mathbb{R}^n \]
  and Jacobian
  \[ J_{ij}(x) = \left( \frac{\partial e_{ij}(x)}{\partial c_i}, \frac{\partial e_{ij}(x)}{\partial c_j}, \ldots, \frac{\partial e_{ij}(x)}{\partial c_n} \right) \in \mathbb{R}^{6 \times 6n} \]

Derivatives of the Error Terms

- Does one error function \( e_{ij}(x) \) depend on all state variables in \( x \)?
  - No, \( e_{ij}(x) \) depends only on \( c_i \) and \( c_j \)

- Is there any consequence on the structure of the Jacobian?
  - Yes, it will be non-zero only in the columns corresponding to \( c_i \) and \( c_j \)
  - Jacobian is sparse
  \[ J_{ij}(x) = \begin{bmatrix} 0 & \ldots & \frac{\partial e_{ij}(x)}{\partial c_i} & \ldots & \frac{\partial e_{ij}(x)}{\partial c_j} & \ldots & 0 \end{bmatrix} \]
**Linearizing the Error Function**

Linearize \( f(x) = \sum_{ij} e_{ij}(x)^T \Sigma_{ij}^{-1} e_{ij}(x) \)
\( \simeq c + 2b^T \Delta x + \Delta x^T H \Delta x \)

with \( b^T = \sum_{ij} e_{ij}^T \Sigma_{ij}^{-1} J_{ij} \)
\( H = \sum_{ij} J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \)

**(Linear) Least Squares Minimization**

1. Linearize error function
   \( f(x + \Delta x) \simeq c + 2b^T \Delta x + \Delta x^T H \Delta x \)
2. Compute the derivative
   \( \frac{df(x + \Delta x)}{d\Delta x} = 2b + 2H \Delta x \)
3. Set derivative to zero
   \( H \Delta x = -b \)
4. Solve this linear system of equations, e.g.,
   \( \Delta x = -H^{-1}b \)

**Gauss-Newton Method**

**Problem:** \( f(x) \) is non-linear!

**Algorithm:** Repeat until convergence

1. Compute the terms of the linear system
   \( b^T = \sum_{ij} e_{ij}^T \Sigma_{ij}^{-1} J_{ij} \)
   \( H = \sum_{ij} J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \)
2. Solve the linear system to get new increment
   \( H \Delta x = -b \)
3. Update previous estimate
   \( x \leftarrow x + \Delta x \)

**Structure of the Minimization Problem**

Linearize \( f(x) = \sum_{ij} e_{ij}(x)^T \Sigma_{ij}^{-1} e_{ij}(x) \)
\( \simeq c + 2b^T \Delta x + \Delta x^T H \Delta x \)

with \( b^T = \sum_{ij} e_{ij}^T \Sigma_{ij}^{-1} J_{ij} \in \mathbb{R}^{6n} \)
\( H = \sum_{ij} J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \in \mathbb{R}^{6n \times 6n} \)

- What is the structure of \( b^T \) and \( H \)?
  (Remember: all \( J_{ij} \)'s are sparse)

**Illustration of the Structure**

- Non-zero only at \( c_i \) and \( c_j \)

- Non-zero only at \( c_i \) and \( c_j \)

- Non-zero on the main diagonal at \( c_i \) and \( c_j \)
Illustration of the Structure

\[ b_{ij}^T = e_{ij}^T \Sigma^{-1} J_{ij} \]

\[ H_{ij} = J_{ij}^T \Sigma^{-1} J_{ij} \]

Sparsity of the Hessian

- Remember: We have to solve \( H \Delta x = -b \)
- The Hessian is
  - positive semi-definite
  - symmetric
  - sparse
- This allows the use of efficient solvers
  - Sparse Cholesky decomposition (~100M matrix elements)
  - Preconditioned conjugate gradients (~1.000M matrix elements)
  - ... many others

Example in 1D

- Two camera poses \( c_1, c_2 \in \mathbb{R} \)
- State vector \( x = (c_1, c_2)^T \in \mathbb{R}^2 \)
- One (distance) observation \( z_{12} \in \mathbb{R} \)
- Initial guess \( c_1 = c_2 = 0 \)
- Observation \( z_{12} = 1 \)
- Sensor noise \( \Sigma_{12} = 0.5 \)

What Went Wrong?

- The constraint only specifies a relative constraint between two nodes
- Any poses for the nodes would be fine as long as their relative pose fits
- One node needs to be fixed
  - Option 1: Remove one row/column corresponding to the fixed pose
  - Option 2: Add to \( H \), \( b \) a linear constraint \( 1 \cdot \Delta c_1 = 0 \)
  - Option 3: Add the identity matrix to \( H \) (Levenberg-Marquardt)
Fixing One Node

- The constraint only specifies a **relative constraint** between two nodes.
- Any poses for the nodes would be fine as long as their relative pose fits.
- One node needs to be fixed (here: Option 2).

\[
\begin{align*}
H &= \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \\
\Delta x &= -H^{-1}b \\
\Delta x &= (0 \ 1)^T
\end{align*}
\]

Additional constraint that sets \( \Delta \alpha = 0 \)

Levenberg-Marquardt Algorithm

- **Observations:**
  - Gauss-Newton method typically converges very quickly.
  - Sometimes diverges when initial solution is far off.
  - Gradient descent (with line search) never diverges.
- How can we combine the advantages of both minimization methods?

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor
  \[
  (H + \lambda I)\Delta x = -b \\
  (J^T J + \lambda I)\Delta x = -J^T e
  \]
- What is the effect of this damping factor?
  - Small \( \lambda \) \( \rightarrow \) same as least squares
  - Large \( \lambda \) \( \rightarrow \) steepest descent (with small step size)
- **Algorithm**
  - If error decreases, accept \( \Delta x \) and reduce \( \lambda \)
  - If error increases, reject \( \Delta x \) and increase \( \lambda \)

Non-Linear Minimization

- One of the state-of-the-art solution to compute the maximum likelihood estimate.
- Various open-source implementations available:
  - g2o [Kuemmerle et al., 2011]
  - sba [Lourakis and Argyros, 2009]
  - iSAM [Kaess et al., 2008]
  - Ceres [Google, 2012]
- Other extensions:
  - Robust error functions
  - Alternative parameterizations

Example: Google Street View

Map Optimization with Ceres Solver

[Google, 2012]

Example: RGB-D SLAM

[Engelhard et al., 2011; Endres et al., 2012]

- **Given:** Kinect data
  - Color image
  - Registered depth image (=point cloud)
- **Wanted:**
  - Camera poses
  - Aligned point cloud
Example: RGB-D SLAM
[Engelhard et al., 2011; Endres et al., 2012]

- Step 1: Extract 2D features (SIFT)

Lessons Learned Today

- How to separate inliers from outliers using RANSAC
- How to align point clouds using ICP
- How to model the SLAM problem in a graph
- How to optimize the map using non-linear least squares

Visual Navigation for Flying Robots

Bundle Adjustment and Dense 3D Reconstruction

Dr. Jürgen Sturm
Agenda for Today

- Bundle adjustment
- Depth cameras
- Occupancy grid maps
- Signed distance functions

Reminder: Pose Graph SLAM

- **Given:** Set of relative pose observations $z_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $c_1, \ldots, c_n \in \mathbb{R}^6$

Reminder: Pose Graph SLAM

- **Goal:** Minimize the error over all constraints
  \[ x^* = \arg \min_x \sum_{ij} e_{ij}(x) \Sigma_{ij}^{-1} e_{ij}(x) \]

Bundle Adjustment

- **Graph SLAM:** Optimize (only) the camera poses
  \[ x = (c_1^T, \ldots, c_n^T)^T \in \mathbb{R}^{6n} \]

- **Bundle Adjustment:** Optimize both 6DOF camera poses and 3D (feature) points
  \[ x = (c_1^T, \ldots, c_n^T, p_1^T, \ldots, p_m^T)^T \in \mathbb{R}^{6n+3m} \]
  - Typically $m \gg n$ (why?)

Error Function

- Camera pose $c_i \in \mathbb{R}^6$
- Feature point $p_j \in \mathbb{R}^3$
- Observed feature location $z_{ij} \in \mathbb{R}^2$
- Expected feature location
  \[ g(c_i, p_j) = R_i^T(t_i - p_j) \]
  \[ h(c_i, p_j) = g_xg(c_i, p_j) / g_z(c_i, p_j) \]
**Error Function**

- Difference between observation and expectation
\[ e_{ij} = z_{ij} - h(c_i, p_j) \]
- Error function
\[ f(c, p) = \sum_{ij} e_{ij}^T \Sigma^{-1} e_{ij} \]
- Covariance \( \Sigma \) is often chosen isotropic and on the order of one pixel

---

**Primary Structure**

- Characteristic structure
\[
\begin{pmatrix}
J^T_c J_c & J^T_c J_p \\
J^T_p J_c & J^T_p J_p
\end{pmatrix}
\begin{pmatrix}
\Delta c \\
\Delta p
\end{pmatrix}
= \begin{pmatrix}
-J^T_c e_c \\
-J^T_p e_p
\end{pmatrix}
\]
\[
\begin{pmatrix}
H_{cc} & H_{cp} \\
H_{pc} & H_{pp}
\end{pmatrix}
\begin{pmatrix}
\Delta c \\
\Delta p
\end{pmatrix}
= \begin{pmatrix}
-J^T_c e_c \\
-J^T_p e_p
\end{pmatrix}
\]

---

**Schur Complement**

- Given: Linear system
\[
\begin{pmatrix}
A & B \\
C & D
\end{pmatrix}
\begin{pmatrix}
x \\
y
\end{pmatrix}
= \begin{pmatrix}
a \\
b
\end{pmatrix}
\]
- If \( D \) is invertible, then (using Gauss elimination)
\[
(A - BD^{-1}C)x = a - BD^{-1}b \\
y = D^{-1}(b - Cx)
\]
- Reduced complexity, i.e., invert one \( p \times p \) and \( p \times p \) matrix instead of one \( (p + q) \times (p + q) \) matrix

---

**Example Hessian**

(Lourakis and Argyros, 2009)

\[
H = \begin{pmatrix}
\end{pmatrix}
\]

---

**Two Examples**

- **PTAM**
http://www.robots.ox.ac.uk/~gk/publications/KleinMurray2007ISMAR.pdf

- **Photo Tourism**
http:// phototour.cs.washington.edu/Photo_Tourism.pdf
PTAM (2007)

- Architecture optimized for dual cores
- Tracking thread runs in real-time (30Hz)
- Mapping thread is not real-time

PTAM – Feature Tracking

- Generate 8x8 matching template (warped from key frame to current pose estimate)
- Search a fixed radius around projected position
  - Using SSD
  - Only search at FAST corner points

PTAM – Example Timings

- Tracking thread
  - Total: 19.2 ms
    - Key frame preparation: 2.2 ms
    - Feature Projection: 3.5 ms
    - Patch search: 9.8 ms
    - Iterative pose update: 3.7 ms

- Mapping thread
  - Key frames 2-49:
    - Local Bundle Adjustment: 170 ms, 270 ms, 440 ms
    - Global Bundle Adjustment: 380 ms, 1.7 s, 6.9 s

PTAM Video

Parallel Tracking and Mapping for Small AR Workspaces

Extra video results made for ISMAR 2007 conference

Georg Klein and David Murray
Active Vision Laboratory
University of Oxford
Photo Tourism (2006) / Bundler

- Overview

- Input Photographs (from Flickr)
- Scene Reconstruction
- Photo Explorer

- Relative camera positions and orientations
- Point cloud
- Sparse correspondence

Photo Tourism – Scene Reconstruction

- Overview

- Processing pipeline
  - Feature detection (SIFT)
  - Pair-wise matching
  - Correspondence estimation (RANSAC)
  - Incremental structure from motion

- Automatically estimate
  - Position, orientation and focal length of all cameras
  - 3D positions of point features

Photo Tourism – Input Images

- Feature detection (SIFT)
- Pair-wise matching
- Correspondence estimation (RANSAC)
- Incremental structure from motion

Photo Tourism – Feature Detection

- Feature matching

Photo Tourism – Feature Matching

- Incremental Structure From Motion

- To help get good initializations, start with two images only (compute pose, triangulate points)
- Non-linear optimization
- Iteratively add more images
**Photo Tourism – Video**

*Photo Tourism*
Exploring photo collections in 3D

Noah Snavely  Steven M. Seitz  Richard Szeliski
University of Washington  Microsoft Research
SIGGRAPH 2006

**From Sparse Maps to Dense Maps**

- So far, we only looked at sparse 3D maps
  - We know where the (sparse) cameras are
  - We know where the (sparse) 3D feature points are
- How can we turn these models into volumetric 3D models?

**Human Stereo Vision**

**Stereo Correspondence Constraints**

- Given a point in the left image, where can the corresponding point be in the right image?

**Reminder: Epipolar Geometry**

- A point in one image “generates” a line in another image (called the epipolar line)
- Epipolar constraint $\hat{x}_2^T E \hat{x}_1 = 0$

**Epipolar Constraint**

- This is useful because it reduces the correspondence problem to a 1D search along an epipolar line
Example: Converging Cameras

Example: Parallel Cameras

Rectification

- In practice, it is convenient if the image scanlines (rows) are the epipolar lines
  - Reproject image planes onto a common plane parallel to the baseline (two 3x3 homographies)
- Afterwards pixel motion is horizontal

Basic Stereo Algorithm

- For each pixel in the left image
  - Compare with every pixel on the same epipolar line in the right image
  - Pick pixel with minimum matching cost (noisy)
  - Better: match small blocks/patches (SSD, SAD, NCC)

Block Matching Algorithm

**Input:** Two images and camera calibrations

**Output:** Disparity (or depth) image

**Algorithm:**
1. Geometry correction (undistortion and rectification)
2. Matching cost computation along search window
3. Extrema extraction (at sub-pixel accuracy)
4. Post-filtering (clean up noise)
What is the Influence of the Block Size?

- Common choices are 5x5 .. 11x11
- Smaller neighborhood: more details
- Larger neighborhood: less noise
- Suppress pixels with low confidence (e.g., check ratio best match vs. 2nd best match)

Problems with Stereo

- Block matching typically fails in regions with low texture
  - Global optimization/regularization (speciality of our research group)
  - Additional texture projection

Sensor Principle of Kinect

- Kinect projects a diffraction pattern (speckles) in near-infrared light
- CMOS IR camera observes the scene

Example Data

- Kinect provides color (RGB) and depth (D) video
- This allows for novel approaches for (robot) perception
**Sensor Principle of Kinect**

- Pattern is memorized at a known depth
- For each pixel in the IR image
  - Extract 9x9 template from memorized pattern
  - Correlate with current IR image over 64 pixels and search for the maximum
  - Interpolate maximum to obtain sub-pixel accuracy (1/8 pixel)
- Calculate depth by triangulation

**Technical Specs**

- Infrared camera has 640x480 @ 30 Hz
- Depth correlation runs on FPGA
- 11-bit depth image
- 0.8m – 5m range
- Depth sensing does not work in direct sunlight (why?)
- RGB camera has 640x480 @ 30 Hz

**Impact of the Kinect Sensor**

- Sold >18M units
- Has become a “standard” sensor in robotics
- Several variants (Asus Xtion Pro, PrimeSense)

**Time-of-Flight Cameras**

- Direct time-of-flight measurement
  - Emit short light pulse (flash)
  - Every pixel counts time until signal is detected

- Indirect measurement (phase shift)
  - Emit modulated light (e.g., at 30 MHz \( \rightarrow \) 10m wavelength)
  - Every pixel measures the phase shift
Decoding the Phase

- Take four intensity measurements at 90° angle
- Integrate over several waves to reduce noise
- Decode amplitude, offset, phase shift

![Intensity measurements diagram](image)

\[
A = \frac{1}{2} \sqrt{(A_3 - A_1)^2 + (A_2 - A_0)^2} \\
B = A_0 + A_1 + A_2 + A_3 \\
\phi = \arctan \frac{A_3 - A_1}{A_2 - A_0}
\]

![Intensity measurements diagram](image)

Commercial Time-Of-Flight Sensors

- Mesa SwissRanger ($4300), 176x144
- PMDTec: 200x200
- Intel Creative Camera ($150), 320x240
- Xbox One Kinect, 512x424, 13-bit depth

![Time-of-flight sensors](image)

Intel Perceptual Computing Challenge

- [https://perceptualchallenge.intel.com/](https://perceptualchallenge.intel.com/)
- API for raw data + hand gesture recognition
- This Saturday (22.6.2013): Hacknight
  - Every participating team gets a sensor for free
  - WERK1, Kultfabrik, Grafinger Str. 6, 81671 München
  - Sign up on Facebook page: [http://www.facebook.com/groups/154028434769715/](http://www.facebook.com/groups/154028434769715/)
- Cash Prizes for the Hacknight
  - €2.500 for the Best app
  - €1.000 for the Most Innovative app
  - €1.000 for the Best User Experience app

![Intel Perceptual Computing Challenge](image)

Project with Ardrones?

- Control your Parrot AR.Drone using:
  - gestures to take-off and land
  - your face and your bare hands to steer the drone in all directions
  - speech for special flight animations and commands
- fly races with friends without touching any controller
- fly through obstacle courses and test your skillfulness
- AR.Drone video transmission and time-races included
- no 3D graphics card needed – it’s real!
Agenda for Today

- Bundle adjustment ✓
- Depth cameras ✓
- Occupancy grid maps
- Signed distance functions

Mapping and 3D Reconstruction

- So far: We have camera poses and 3D points

- Robot needs a map for:
  - Path planning and collision-free navigation
  - Exploration of unmapped areas
  - How can we estimate such a map?

Occupancy Grid

Idea:
- Represent the map \( m \) using a grid
- Each cell is either free or occupied
- Map is static
- Cells have binary state (empty or occupied)
- All cells are independent of each other
- Robot maintains a belief \( \text{Bel}(m) \) on map state
- As a result, each cell \( m_i \) can be estimated independently from the sensor observations
- Will also drop index \( i \) (for the moment)

Goal: Estimate the belief from sensor observations

\[
\text{Bel}(m) = P(m | z_1, \ldots, z_n)
\]

Mapping

Goal: Estimate

\[
\text{Bel}(m) = P(m | z_1, \ldots, z_n)
\]

How can this be computed?

Binary Bayes Filter

Goal: Estimate

\[
\text{Bel}(m) = P(m | z_1, \ldots, z_n)
\]

How can this be computed?

Using the (binary) Bayes Filter from Lecture 3

\[
P(m | z_{1:t}) = \left( 1 + \frac{1 - P(m | z_t) P(m | z_{1:t-1})}{P(m | z_t) P(m | z_{1:t-1}) - 1 - P(m)} \right)^{-1}
\]
Binary Bayes Filter

- Prior probability that cell is occupied $P(m)$ (often 0.5)
- Inverse sensor model $P(m \mid z_i)$ is specific to the sensor used for mapping
- The log-odds representation can be used to increase speed and numerical stability

\[ L(x) := \log \frac{p(x)}{p(-x)} = \log \frac{p(x)}{1 - p(x)} \]

Binary Bayes Filter using Log-Odds

- In each time step, compute

\[ L(m \mid z_{1:t}) = L(m \mid z_{1:t-1}) + L(m \mid z_t) + L(m) \]

- When needed, compute current belief as

\[ \text{Bel}_t(m) = 1 - \frac{1}{1 + \exp L(m \mid z_{1:t})} \]

Clamping Update Policy

- Often, the world is not “fully” static
- Consider an appearing/disappearing obstacle
- To change the state of a cell, the filter needs as many positive (negative) observations
- Idea: Clamp the beliefs to min/max values

\[ L'(m \mid z_{1:t}) = \max(\min(L(m \mid z_{1:t}), l_{\text{max}}), l_{\text{min}}) \]

Sensor Model

- For the Bayes filter, we need the inverse sensor model $p(m \mid z)$

- Let’s consider an ultrasound sensor
  - Located at (0,0)
  - Measures distance of 2.5m
- How does the inverse sensor model look like?

Typical Sensor Model for Ultrasound

- Combination of a linear function (in x-direction) and a Gaussian (in y-direction)

- Question: What about a laser scanner?
**Resulting Map**

![Map](image)

**Note:** The maximum likelihood map is obtained by clipping the occupancy grid map at a threshold of 0.5

---

**Memory Consumption**

- Consider we want to map a building with 40x40m at a resolution of 0.05cm
- How much memory do we need?

\[
\left(\frac{40}{0.05}\right)^2 = 640,000 \text{ cells} = 4.88 \text{mb}
\]

- And for 3D?

\[
\left(\frac{40}{0.05}\right)^3 = 512,000,000 \text{ cells} = 3.8 \text{gb}
\]

- And what about a whole city?

---

**Map Representation by Octtrees**

- Tree-based data structure
- Recursive subdivision of space into octants
- Volumes can be allocated as needed
- Multi-resolution

---

**Example: OctoMap**

[Wurm et al., 2011]

- Freiburg, building 79
  - 44 x 18 x 3 m³, 0.05m resolution, 0.7mb on disk

---

**Example: OctoMap**

[Wurm et al., 2011]

- Freiburg computer science campus
  - 292 x 167 x 28 m³, 0.2m resolution, 2mb on disk
**Signed Distance Field (SDF)**

- **Idea:** Instead of representing the cell occupancy, represent the distance of each cell to the surface.
- **Occupancy grid maps:** explicit representation

SDF: implicit representation

\[
\begin{array}{c|ccccc}
  & 0 & 1 & 0.5 & 0.5 & 1.8 \\
\hline
\text{zero} & \text{free space} \\
\text{negative} & \text{outside obj.} \\
\text{positive} & \text{inside obj.} \\
\text{one} & \text{occupied} \\
\end{array}
\]

Signed Distance Field (SDF) \[\text{[Curless and Levoy, 1996]}\]

**Algorithm:**
1. Estimate the signed distance field
2. Extract the surface using interpolation (surface is located at zero-crossing)

**Distance and Weighting Functions**

- Weight each observation according to its confidence
- Weight can additionally be influenced by other modalities (reflectance values, ...)

**Dense Mapping: 2D Example**

- Camera with known pose
- For each grid cell, compute its projective distance to the surface
Dense Mapping: 3D Example

- Generalizes directly to 3D
- But: memory usage is cubic in side length

Data Fusion

- **Idea:** Compute weighted average
- Each voxel cell \( x \) in the SDF stores two values
  - Weighted sum of signed distances \( D_t(x) \)
  - Sum of all weights \( W_t(x) \)
- When new range image arrives, update every voxel cell according to
  \[
  D_{t+1}(x) = D_t(x) + w_{t+1}(x)d_{t+1}(x) \\
  W_{t+1}(x) = W_t(x) + w_{t+1}(x)
  \]

Two Nice Properties

- Noise cancels out over multiple measurements
- Zero-crossing can be extracted at sub-voxel accuracy (least squares estimate)
  \[
  1D \text{ Example: } x^* = \frac{\sum D_t(x)x}{\sum W_t(x)x}
  \]

Visualizing Signed Distance Fields

Common approaches to iso surface extraction:

1. **Ray casting** (GPU, fast)
   - For each camera pixel, shoot a ray and search for zero crossing
2. **Polygonization** (CPU, slow)
   - E.g., using the marching cubes algorithm
   - Advantage: outputs triangle mesh

Ray Casting

- For each camera pixel, shoot a ray and search for the first zero crossing in the SDF
- Value in the SDF can be used to skip along when far from surface

Marching Cubes

First in 2D, **marching squares**:

- Evaluate each cell separately
- Check which edges are inside/outside
- Generate triangles according to lookup table
- Locate vertices using least squares

Visual Navigation and 3D Reconstruction
Marching Cubes

KinectFusion
[Newcombe et al., 2011]
- Projective ICP with point-to-plane metric
- Truncated signed distance function (TSDF)
- Ray Casting

Dense Tracking: 2D Example
[Bylow et al., RSS 2013]
- Minimize distance between depth image and SDF

Dense Tracking: 2D Example
[Bylow et al., RSS 2013]
- Minimize distance between depth image and SDF
Dense Tracking: 2D Example
[Bylow et al., RSS 2013]

- Minimize distance between depth image and SDF

\[ \arg \min_{\xi} E(\xi) = \arg \min_{\xi} \frac{1}{M} \sum_{ij} V(X(\xi, (i, j), I_d))^2 \]

3D Reconstruction from a Quadrocopter
[Bylow et al., RSS 2013]

- AscTec Pelican quadrocopter
- Real-time 3D reconstruction, position tracking and control

Resulting 3D Model
[Bylow et al., RSS 2013]

Let’s Scan a Person!
[Sturm et al., GCPR 2013]

3D Color Printing

Can We Print These Models in 3D?

- Who wants to get a 3D scan of him/herself?
Lessons Learned Today

- How to estimate the camera poses and 3D points from monocular images using bundle adjustment
- How depth cameras work
- How to estimate occupancy maps
- What signed distance functions are
- How to reconstruct triangle meshes from SDFs

AR.Drone: Visual Navigation

How to make the AR.Drone fly more stable?

1. time delays
2. marker → PTAM

1. Time Delays

capture frame → send to PC → compute → send control
takes 150ms - 250ms

Solution: Explicitly model delays

1. Time Delays

Solution: Model control & dynamics

\[ \text{control} \rightarrow \text{attitude} \quad \text{attitude} \rightarrow \text{velocity} \]

\[ \rightarrow \text{enables fully “blind” prediction} \]

Full EKF

\[ h_{\text{PTAM}}(x) := (x, y, z, \Phi, \Theta, \Psi) \in \mathbb{R}^6 \]

\[ z_{\text{PTAM}} := \log(E_T^T E_C) \in \mathbb{R}^6 \]

PTAM Observation

IMU, altimeter, velocity Observation

2. PTAM

Parallel Tracking and Mapping [Murray ’07]:

\[ \rightarrow \text{keyframe based, monocular SLAM system} \]

\[ \rightarrow \text{open-source} \]

Problems: Unreliable, no scale

\[ \rightarrow \text{enhance reliability by incorporating IMU data} \]

\[ \rightarrow \text{add scale-estimation from altimeter} \]

Start and Initialize Map
**tum_ardrone ROS package**

Node 1: GUI + backup control

Node 2: State-estimation (SLAM, EKF)

Node 3: Autopilot (PID Controller)

Open Source @ www.ros.org/wiki/tum_ardrone

**Future Work**

- **Increase Range:**
  Augment PTAM with eg. FABMAP and/or g2o

- **Initialization:**
  Faster & more robust using gyro readings;
  Automatic re-initialization

- **Performance (→ onboard):**
  e.g. only 3DoF coarse tracking + gyro.

**Future Work**

- **Add feature**
  e.g. person following, gesture recognition, …

- **Obstacle Avoidance**
  e.g. using optical flow.

- ...

**Preview**

Monocular SLAM without keypoints

- Image (pixels)
- Features (e.g. point-positions)
- Mapping, Tracking

**Preview**

Monocular SLAM without keypoints

- Image (pixels)
- Features (e.g. point-positions)
- Mapping, Tracking

-?
Questions?

Dense Visual SLAM for RGB-D cameras

Christian Kerl, Jürgen Sturm, and Daniel Cremers

Goal

Estimate camera motion from RGB-D data

Application Domains

- Position control
- Autonomous navigation
- 3D reconstruction

Dense Visual Odometry

\[ p = \pi^{-1}(x, Z_1(x)) \quad p' = T_\xi p \quad x' = \pi(p') \]

- Photometric consistency
  \[ I_2(x') = I_1(x) \]
- Geometric consistency
  \[ Z_2(x') = p'_z \]
Dense Visual Odometry

- Least squares formulation

\[
    \mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} I_2(x') - I_1(x) \\ Z_2(x') - p_z' \end{pmatrix}
\]

\[
    \xi^* = \arg \min_{\xi} \sum_i^n \mathbf{e}_i^T \Sigma^{-1} \mathbf{e}_i
\]

Robust Dense Visual Odometry

- Outliers violate consistency assumption
  - Moving objects
  - Non-lambertian surfaces
  - Noise
- Problem: Quadratic term gives high influence

Robust Dense Visual Odometry

- Weighted least squares formulation

\[
    \xi^* = \arg \min_{\xi} \sum_i^n w_i \mathbf{e}_i^T \Sigma^{-1} \mathbf{e}_i
\]

\[
    w_i(\mathbf{e}_i) = \frac{\nu+1}{\nu + \mathbf{e}_i^T \Sigma^{-1} \mathbf{e}_i}
\]
Visual Odometry Results

- Frame-to-frame motion estimation
  - Fast
  - Highly accurate
  - Drift 0.03 m/s

- Problem: drift accumulation (1.8 m/min)

Dense Visual SLAM

- Local drift
  - Keyframes

- Global drift
  - Pose graph optimization

Keyframes

- Frame-to-frame

Pose Graph Optimization

- Correct global drift with loop closures
### Pose Graph Optimization
- Search for loop closure candidates

### Pose Graph Optimization
- Validate loop closure and update graph

### Dense Visual SLAM
- How to select keyframes?
- How to validate loop closures?

### Dense Visual SLAM
- Least squares yields estimate of covariance of $\xi^*$
- Compute entropy of parameter distribution as $H(\xi) = \ln(|\Sigma_\xi|)$
- $H(\xi)$ is a measure of uncertainty in estimate, i.e., quality

### Visual SLAM Results

### Master Thesis Topics
- Dense Visual SLAM for Quadrocopters
  » Implement on AscTec Pelican
- Multi-Session Dense Visual SLAM
  » Relocalization / place recognition
  » Reduced pose graph
  » Efficient map representation
Visual Navigation for Flying Robots

Dense Reconstruction in Sparse Data Structures

Frank Steinbrücker

Recap: Signed Distance Field (SDF)  
[Curless and Levoy, 1996]

- Idea: Instead of representing the cell occupancy, represent the distance of each cell to the surface
- Occupancy grid maps: explicit representation
- SDF: implicit representation

Recap: Dense Mapping: 2D Example

- Camera with known pose
- For each grid cell, compute its projective distance to the surface

Algorithm:
1. Estimate the signed distance field
2. Extract the surface using interpolation (surface is located at zero-crossing)
Recap: Dense Mapping: 3D Example

- Generalizes directly to 3D
- But: memory usage is cubic in side length

Recap: Data Fusion

- Idea: Compute weighted average
- Each voxel cell \( x \) in the SDF stores two values
  - Weighted sum of signed distances \( D_t(x) \)
  - Sum of all weights \( W_t(x) \)
- When new range image arrives, update every voxel cell according to

\[
D_{t+1}(x) = D_t(x) + w_{t+1}(x)d_{t+1}(x) \\
W_{t+1}(x) = W_t(x) + w_{t+1}(x)
\]

Recap: Distance and Weighting Functions

- Weight each observation according to its confidence
- Weight can additionally be influenced by other modalities (reflectance values, …)

Recap: Two Nice Properties

- Noise cancels out over multiple measurements
- Zero-crossing can be extracted at sub-voxel accuracy (least squares estimate)

1D Example: 

\[
x^* = \frac{\sum D_t(x)x}{\sum W_t(x)x}
\]

Recap: Memory Consumption

- Consider we want to map a building with 40x40m at a resolution of 0.05m
- How much memory do we need?

\[
(\frac{40}{0.05})^2 = 640.000 \text{ cells} = 4.88\text{mb}
\]
- And for 3D?

\[
(\frac{40}{0.05})^3 = 512.000.000 \text{ cells} = 3.8\text{gb}
\]
- And what about a whole city?
**Distance and Weighting Functions**

- Values outside the support of the weight function do not need to be stored

- \[ \text{weight } w(x) \quad (= \text{confidence}) \]

- \[ \text{truncated signed distance to surface } d(x) \]

- \[ \text{measured depth } \]

- distance \[ x \]

**Reconstruction Band around the Surface**

- Values outside the support of the weight function do not need to be stored

- Requirements
  - Low storage cost
  - Time-Efficient update of the band
  - Online-Capability, i.e. not all data is present beforehand
    - Volume "grows" when more depth images are added
    - Only bricks in the current camera frustum are touched

- **Structure 1: Brick List/Array**
  - Every brick stores a position in space
    - Low storage cost => O(1)
    - Time-Efficient update of the band => O(n)
    - Online-Capability, i.e. not all data is present beforehand
      - Volume "grows" when more depth images are added
      - Only bricks in the current camera frustum are touched

- **Structure 2: Brick Grid**
  - Coarse Grid stores occupancy of the bricks
    - Low storage cost => O(n)
    - Time-Efficient update of the band => O(1)
    - Online-Capability, i.e. not all data is present beforehand
      - Volume "grows" when more depth images are added
      - Only bricks in the current camera frustum are touched
Structure 3: Tree and List

- Brick List/Array with an Octree on top
  - Low storage cost $\Rightarrow O(\log n)$ ✓
  - Time-efficient update of the band $\Rightarrow O(\log n)$ ✓
  - Online-Capability, i.e. not all data is present beforehand
    - Volume "grows" when more depth images are added ✓
    - Only bricks in the current camera frustum are touched ✓

Fusion step in the Octree

- (Parallel) iteration over all valid depth pixels
  - Bounding box around the 3D point according to distance/weight-threshold
    - Intersection of the bounding box with tree leaves
      - Allocation of new leaves, where necessary
    - Addition of the intersected leaves to a queue
  - (Parallel) processing of the queue (projection and distance update)

Implementation of the Tree

- CPU/GPU interoperability requirements
  - No explicit C-Pointers $\Rightarrow$ Indices instead
  - All memory is allocated beforehand
    - Push-back operations are performed with global indices
  - Minimization of diverging code
    - $\Rightarrow$ Reason for queue

Implementation of the Tree

- Branch array stores indices of leaves and subbranches
  - Leaf array stores position, scale, and queue index
  - Queue stores leaf number

Demo Video

Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences

ICCV 2013 Submission 1688
Motion Planning Problem

What are good performance metrics?

- Execution speed / path length
- Energy consumption
- Planning speed
- Safety (minimum distance to obstacles)
- Robustness against disturbances
- Probability of success
- ...

Motion Planning Examples

Motion planning is sometimes also called the piano mover’s problem
**Agenda for Today**
- Configuration spaces
- Roadmap construction
- Search algorithms
- Path optimization and re-planning
- Path execution

**Configuration Space**
- Work space
  - Position in 3D → 3 DOF
- Configuration space
  - Reduced pose (position + yaw) → 4 DOF
  - Full pose → 6 DOF
  - Pose + velocity → 12 DOF
  - Joint angles of manipulation robot
  - ...
- Planning takes place in configuration space

**Configuration Space**
- The configuration space (C-space) is the space of all possible configurations
- C-space topology is usually not Cartesian
- C-space is described as a topological manifold

**Notation**
- Configuration space $C \subset \mathbb{R}^d$
- Configuration $q \in C$
- Free space $C_{\text{free}}$
- Obstacle space $C_{\text{obs}}$

**Properties**
- $C_{\text{free}} \cup C_{\text{obs}} = C$
- $C_{\text{free}} \cap C_{\text{obs}} = \emptyset$

**Free Space Example**
- What are admissible configurations for the robot? Equiv.: What is the free space?
  - “Point” robot

**Example**
- What are admissible configurations for the robot? Equiv.: What is the free space?
  - “Point” robot
What are admissible configurations for the robot? Equiv.: What is the free space?

Circular robot

- Free configuration space is obtained by sliding the robot along the edge of the obstacle regions "blowing them up" by the robot radius.
- This operation is called the Minowski sum:
  \[ A \oplus B = \{a + b \mid a \in A, b \in B\} \]
  where \( A, B \subseteq \mathbb{R}^d \).

Example: Minowski Sum

- Triangular robot and rectangular obstacle

Computing the Free Space

- C-space is obtained by sliding the robot along the edge of the obstacle regions.
Basic Motion Planning Problem

- **Given**
  - Free space \( C_{\text{free}} \)
  - Initial configuration \( q_I \)
  - Goal configuration \( q_G \)

- **Goal**: Find a continuous path
  \[ \tau : [0, 1] \rightarrow C_{\text{free}} \]
  with \( \tau(0) = q_I \), \( \tau(1) = q_G \)

Motion Planning Sub-Problems

1. **C-Space discretization**
   (generating a graph / roadmap)
2. **Search algorithm**
   (Dijkstra’s algorithm, A*, ...)
3. **Re-planning**
   (D*, ...)
4. **Path tracking**
   (PID control, potential fields, funnels, ...)

C-Space Discretizations

- **Combinatorial planning**
  - Find a solution when one exists (complete)
  - Require polygonal decomposition
  - Become quickly intractable for higher dimensions

- **Sampling-based planning**
  - Weaker guarantees but more efficient
  - Need only point-wise evaluations of \( C_{\text{free}} \)
  - We will have a look at: grid decomposition, road maps, random trees

Grid Decomposition

- Construct a regular grid
- Determine status of every cell (free/occ)
- Simple, but not efficient (why?)
- Not exact (why?)

Probabilistic Roadmaps (PRMs) [Kavraki et al., 1992]

- Grids do not scale well to high dimensions
- Sampling-based approach
- **Vertex**: Take random sample from \( C \), check whether sample is in \( C_{\text{free}} \)
Probabilistic Roadmaps (PRMs)  
[Kavraki et al., 1992]  
- **Vertex:** Take random sample from $C$, check whether sample is in $C_{\text{free}}$  
- **Edge:** Check whether line-of-sight between two nearby vertices is collision-free  
- Options for “nearby”: k-nearest neighbors or all neighbors within specified radius  
- Add vertices and edges until roadmap is dense enough

PRM Example  
1. Sample vertex  
2. Find neighbors  
3. Add edges

Rapidly Exploring Random Trees  
[Lavalle and Kuffner, 1999]  
- **Idea:** Grow a tree from start to goal location  
- **Algorithm**  
  1. Initialize tree with first node $q_I$  
  2. Pick a random target location (every 100th iteration, choose $q_G$)  
  3. Find closest vertex in roadmap  
  4. Extend this vertex towards target location  
  5. Repeat steps until goal is reached  
- Why not pick $q_G$ every time?

Rapidly Exploring Random Trees  
- **Algorithm**  
  1. Initialize tree with first node $q_I$  
  2. Pick a random target location (every 100th iteration, choose $q_G$)  
  3. Find closest vertex in roadmap  
  4. Extend this vertex towards target location  
  5. Repeat steps until goal is reached  
- Why not pick $q_G$ every time?  
  - This will fail and run into $C_{\text{obs}}$ instead of exploring
Rapidly Exploring Random Trees
[Lavalle and Kuffner, 1999]

- **RRT**: Grow trees from start and goal location towards each other, stop when they connect

![RRT diagram](image)

RRT Examples

- 2-DOF example
- 3-DOF example (2D translation + rotation)

Non-Holonomic Robots

- Some robots cannot move freely on the configuration space manifold
- Example: A car can not move sideways
  - 2-DOF controls (speed and steering)
  - 3-DOF configuration space (2D translation + rotation)

Example: Blimp Motion Planning
[Müller et al., IROS 2011]

Advantages
- Low power consumption
- Safe navigation capabilities

Challenges
- Seriously underactuated (only 3-DOF control)
- Heavily subject to drift
- Requires kinodynamic motion planning

Example: Blimp Motion Planning
[Müller et al., IROS 2011]

- High-level planner: A* in 4D
- Low-level planner: RRT in 12D considering kinodynamic constraints
Example: Blimp Motion Planning
[Müller et al., IROS 2011]

Rapidly Exploring Random Trees
+ Probabilistic. complete
+ Balance between greedy search and exploration
+ Very popular, many extensions
- Metric sensitivity
- Unknown rate of convergence
- Not optimal, not complete

Summary: Sampling-based Planning
- More efficient in most practical problems but offer weaker guarantees
- Probabilistically complete (given enough time it finds a solution if one exists, otherwise, it may run forever)
- Performance degrades in problems with narrow passages

Motion Planning Sub-Problems
1. C-Space discretization (generating a graph / roadmap)
2. Search algorithms (Dijkstra’s algorithm, A*, ...)
3. Re-planning (D*, ...)
4. Path tracking (PID control, potential fields, funnels, ...)

Search Algorithms
- Given: Graph G consisting of vertices and edges (with associated costs)
- Wanted: Find the best (shortest) path between two vertices
- What search algorithms do you know?

Uninformed Search
- Breadth-first
  - Complete
  - Optimal if action costs equal
  - Time and space $O(b^d)$
- Depth-first
  - Not complete in infinite spaces
  - Not optimal
  - Time $O(b^d)$
  - Space $O(bd)$ (can forget explored subtrees)
Example: Dijkstra’s Algorithm

- Extension of breadth-first with arbitrary (non-negative) costs

Informed Search

- Idea
  - Select nodes for further expansion based on an evaluation function \( f(s) \)
  - First explore the node with lowest value
  - What is a good evaluation function?

What is a Good Heuristic Function?

- Choice is problem/application-specific
- Popular choices
  - Manhattan distance (neglecting obstacles)
  - Euclidean distance (neglecting obstacles)
  - Value iteration / Dijkstra (from the goal backwards)

A* Algorithm

- Initialize
  - OPEN = \{start\}, CLOSED = {}
  - \( f(s) = \inf \)
- While goal not in CLOSED
  - Remove vertex \( s \) from OPEN with smallest estimated cost \( f(s) \)
  - Insert \( s \) into CLOSED
- For every successor \( s' \) of \( s \) not yet in CLOSED,
  - Update \( g(s') = \min(g(s'), g(s) + c(s, s')) \)
  - Insert \( s' \) into OPEN

A* search

- Combines path cost with estimated goal distance
  \[ f(s) = g(s) + h(s) \]
- Heuristic function \( h(s) \) has to be
  - Admissible (never over-estimate the true cost)
  - Consistent (satisfies triangle inequality)

A* is optimal (in the number of expanded nodes) and complete (finds a solution if there is one and fails otherwise)
A* Example

- OPEN = \{s1\}
- CLOSED = \{

Start

\[
g(s1) = 0, h(s1) = 1 \]

\[
g(s2) = \infty, h(s2) = 1 \]

\[
g(s3) = \infty, h(s3) = 2 \]

\[
g(s4) = \infty, h(s4) = 2 \]

\[
g(s5) = \infty, h(s5) = 1 \]

\[
g(s6) = \infty, h(s6) = 0 \]

Goal

Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots
Consider the following path planning problem:

- How many states will be expanded by the previous search algorithms?

Dijkstra expands states in the order of $f = g$ values.

- For large problems, this results in $A^*$ quickly running out of memory (many OPEN/CLOSED states).

$A^*$ expands states in the order of $f = g + h$ values.

- For large problems, this results in $A^*$ quickly running out of memory (many OPEN/CLOSED states).

Weighted $A^*$ search expands states in the order of $f = g + \epsilon h$.

- $\epsilon > 1 \rightarrow$ bias towards states that are closer to the goal.

- Search is typically orders of magnitude faster.

- Found path may be longer (by a factor of $\epsilon$).
Anytime A*

- Constructing anytime search based on A*
  - Find the best possible path for a given $\varepsilon$
  - Reduce $\varepsilon$ and re-plan

\[
\begin{array}{ccc}
\text{c=2.5} & \text{c=1.5} & \text{c=1.0} \\
\text{expansions: 13} & \text{expansions: 15} & \text{expansions: 20} \\
\text{moves: 11} & \text{moves: 11} & \text{moves: 10}
\end{array}
\]

Comparison Search Algorithms

Dr. Jürgen Sturm, Computer Vision Group, TUM

Visual Navigation for Flying Robots

63

D* Search

- **Problem:** In unknown, partially known or dynamic environments, the planned path may be blocked and we need to replan.
- Can this be done efficiently, avoiding to replan the **entire path**?

D* Example

- **Initial search**
  - Backwards A* 
  - D* Lite 

- **Second search**
  - Backwards A* 
  - D* Lite 

Main concepts

- **Invert search direction** (from goal to start)
  - Goal does not move, but robot does
  - Map changes (new obstacles) have only local influence close to current robot pose
- **Mark** the changed node and all dependent nodes as **unclean** (= to be re-evaluated)
- **Find shortest path** to start (using A*) while re-using previous solution

D* Search

- **Idea:** Incrementally repair path keeping its modifications local around robot pose
- Many variants:
  - D* (Dynamic A*) [Stentz, ICRA ’94] [Stentz, UCAI ’95]
  - D* Lite [Koenig and Likhachev, AAAI ’02]
  - Field D* [Ferguson and Stenz, JFR ’06]
D* Search

- D* is as optimal and complete as A*
- D* and its variants are widely used in practice
- Field D* was running on Mars rovers Spirit and Opportunity

Problems on A*/D* on Grids

1. The shortest path is often very close to obstacles (cutting corners)
   - Uncertain path execution increases the risk of collisions
   - Uncertainty can come from delocalized robot, imperfect map, or poorly modeled dynamic constraints
2. Trajectories are aligned to grid structure
   - Path looks unnatural
   - Paths are longer than the true shortest path in continuous space

Problems on A*/D* on Grids

3. When the path turns out to be blocked during traversal, it needs to be re-planned from scratch
   - In unknown or dynamic environments, this can occur very often
   - Replanning in large state spaces is costly
   - Can we re-use (repair) the initial plan?

Let’s look at solutions to these problems...

Map Smoothing

- Problem: Path gets close to obstacles
- Solution: Convolve the map with a kernel (e.g., Gaussian)
  - Leads to non-zero probability around obstacles
  - Evaluation function
  \[ f(n) = g(s) \cdot p_{ocf}(s) + h(s) \]

Example: Map Smoothing
Path Smoothing

- **Problem**: Paths are aligned to grid structure (because they have to lie in the roadmap)
- Paths look unnatural and are sub-optimal
- **Solution**: Smooth the path after generation
  - Traverse path and find pairs of nodes with direct line of sight; replace by line segment
  - Refine initial path using non-linear minimization (e.g., optimize for continuity/energy/execution time)

Example: Path Smoothing
- Replace pairs of nodes by line segments
- Non-linear optimization

Real-Time Motion Planning

- What is the maximum time needed to re-plan in case of an obstacle detection?
- What if the robot has to react quickly to unforeseen, fast moving objects?
- Do we really need to re-plan for every obstacle on the way?

Layered Motion Planning

- An approximate **global planner** computes paths ignoring the kinematic and dynamic vehicle constraints (not real-time)
- An accurate **local planner** accounts for the constraints and generates feasible local trajectories in real-time (collision avoidance)
Local Planner

- **Given**: Path to goal (sequence of via points), range scan of the local vicinity, dynamic constraints
- **Wanted**: Collision-free, safe, dynamically feasible, and fast motion towards the goal (or next via point)
- Typical approaches:
  - Potential fields
  - Dynamic window approach

Navigation with Potential Fields

- Treat robot as a particle under the influence of a potential field
- **Pro**: Easy to implement
- **Con**: Suffers from local minima, No consideration of dynamic constraints

Navigation with Funnels

[Choi and Latombe, IROS 1991]

- Different regions of the configuration space need different potential fields
- Compose navigation function from overlapping local potential functions (the so-called funnels)

Dynamic Window Approach

[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

Algorithm:
1. Sample the robot’s control space
2. Simulate each sample for a short period of time
3. Score each sample based on proximity to obstacles, proximity to goal, proximity to global path, speed
4. Pick the highest-scoring control command

Consider a 2DOF planar robot
- All possible speeds of the robot $V_s$
- Forward velocity 0.9 m/s
- Angular velocity $\pm 90$ deg/s

Consider a 2DOF planar robot + 2D environment
- All possible speeds of the robot $V_s$
- Forward velocity 0.9 m/s
- Angular velocity $\pm 90$ deg/s
- Obstacle-free area $V_{safe}$
Dynamic Window Approach
[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Consider additionally dynamic constraints
  - all possible speeds of the robot $V_a$
  - forward velocity
  - current robot speed
  - obstacle-free area $V_a$
  - angular velocity $90\text{deg/s}$
  - $+90\text{deg/s}$

Admissible space $V_a \cap V_n \cap V_d$

Dynamic Window Approach
[Simmons, 96], [Fox et al., 97], [Brock & Khatib, 99]

- Navigation function (potential field)
  $$f(n) = \alpha \cdot \text{vel} + \beta \cdot nf + \gamma \cdot \Delta nf + \delta \cdot \text{goal}$$

Maximizes velocity
Rewards alignment to A* path gradient
Rewards large advances on A* path

Example: Dynamic Window Approach
[Brock and Khatib, ICRA '99]

- Discretize dynamic window and evaluate navigation function (note: window has fixed size = real-time!)
- Find the maximum and execute motion
Problems of DWAs

- DWAs suffer from local minima (need tuning), e.g., robot does not slow down early enough to enter doorway:

- Can you think of a solution?
- **Note:** General case requires global planning

Example: Motion Planning in ROS

- Executive: state machine (`move_base`)
- Global costmap: grid with inflation (`costmap_2d`)
- Global path planner: Dijkstra (Dijkstra, `navfn`)
- Local costmap (`costmap_2d`)
- Local planner: Dynamic window approach (`base_local_planner`)

Lessons Learned Today

- How to sample roadmaps and probabilistic random trees
- How to efficiently compute a path between the start and goal node
- How to update plan efficiently
- How to follow and execute a path in real-time

Lecture:
Autonomous micro aerial vehicles

Friedrich Fraundorfer
Remote Sensing Technology
TU München

Autonomous operation@ETH Zürich
Autonomous operation@ETH Zürich

6x speed

Outline

- MAV system
- Optical flow for pose estimation
- 3D mapping for navigation
- Navigation
  - Frontier based exploration
  - VFH+/State lattice

Sensor and System Setup

Front Stereo Camera
Downward Optical Flow Camera
IMU and Altitude
Low level microcontroller
Linux flight computer

Onboard Electronics

- Computer-On-Module (COTS, 18-27W)
  - microETXexpress industry standard module
    - Intel Core 2 DUO 1.86 GHz / Core i7 2.0 GHz (18-40 W)
- Onboard flight computer (pxCOMex)
  - PXHAWK mainboard design
    - 200g (incl. cooling, 27W Core 2, 40W Core i7)
- Inertial Measurement Unit (pxIMUv2.5)
  - 60 MHz ARM7
  - 3D gyroscope, accelerometer, compass
  - Barometric pressure
  - Kalman filtering and PID controllers
  - Triggers cameras

System Architecture

Autonomous Flight

Stereo camera

→ Stereo & VO

Attitude controller

→ Position controller

→ Planner

IMU

→ OF camera
### Important sensors

- **Stereo camera** (timestamped and synchronized with IMU measurements, max. 30fps)
- **Optical flow camera**

### The optical flow idea

**Assumptions:**
- Planar scene
- Camera plane parallel to ground plane
- Optical flow measures x,y shift in pixel
- Z and metric scale from altitude sensor

#### 640 pixel, but how many meters?

\[
\frac{G_x}{h} = \frac{sx}{f} = \tan\left(\frac{\text{FOV}}{2}\right)
\]

\[
G_x = \frac{sx}{h} = \tan\left(\frac{\text{FOV}}{2}\right)\cdot h
\]

640 pixel, but how many meters?

#### The optical flow idea

\[
I_2 = H \ast H
\]

\[
p_2 = H \ast p_1
\]

\[
H = R + \frac{1}{d} TN^T
\]

\[
I_2 = H \ast H
\]

\[
p_2 = H \ast p_1
\]

\[
H = R + \frac{1}{d} TN^T
\]
The optical flow idea

\[ I_2 = H \cdot I_1 \]
\[ p_2 = H \cdot p_1 \]
\[ H = R + \frac{1}{d} TV^T \]

Optical flow camera - PX4Flow

- Smart camera module
  - 720x480V (60fps), 1880x120V (250fps), 16mm lens
- ARM Cortex M4 (168 MHz, 192 KB RAM, single precision floating point operations)
- MEMS gyroscope (L3GD20)
- Ultrasound sensor
- Outputs speed
- Serial interface (Mavlink)
- ROS node (http://www.ros.org/wiki/px4flow_node)

Optical flow camera - PX4Flow

- 64x64 pixel, 250Hz
- SAD optical flow computation
- 8x8 pixel blocks within a 4x4 pixel search range
- Histogram voting
- Subpixel refinement
- Removal of orientation component
- Outputs speed

Multi-volume occupancy grid

- Multi-volume occupancy grid implementation [Morris 2010].
  - Group sensor readings into continuous vertical volumes which are stored in a 2D grid.
  - Record both positive and negative readings.
  - Models free and occupied space.
Multi-volume occupancy grid

- Update 3D grid with distance measurements
- Downsample range data from stereo / Kinect to a virtual scan.
- Outlier removal and efficient occupancy grid updates.
- Each ray in a virtual scan measures the median distance to the range points falling in an angular interval.

Updating the map:
- For each ray in the virtual scan,
  - Traverse in order the cells intersected by the ray.
  - Insert negative volumes in the cells until we reach the endpoint of the ray at which we insert a positive volume.
  - Merge overlapping volumes (changes densities of volumes)

Occupancy probabilities can be computed for any specific coordinate x,y,z
- Typically one can extract occupancy probabilities for different height planes

Exploration

Frontier based exploration
- Method suitable for exploration in grid map (occupancy grid) environment representation
- Frontiers are boundaries between known (sensed) and unknown (unsensed) area.
- MAV is directed to centroid of frontiers
Frontier based exploration

Vector Field Histogram (VFH)
- Only 1 waypoint at a time
- Fly discrete steps toward goal

State lattice
- State lattice concept [Pivtoraiko et al., 2009] can include mobility constraints
- For the MAV we only want to allow forward motion, because of forward looking cameras
- A state lattice is a discretization of the state space and continuous motion primitives that connect these states with edges (graph structure)
- The MAV flies at a fixed attitude; each node represents a 3D state \([x, y, \theta]\).

Safety Clearance = 0.75m

Grid of all possible discrete states (here x,y)
Possible motions (state transitions)

Grid of all possible discrete states (here x,y)
Graph search

Obstacle detection and avoidance

Lessons learned
- System components for autonomous MAV's
- Egomotion estimation using optical flow
- 3D environment mapping using multi-volume occupancy grids
- Frontier based exploration
- Local navigation with VFH and state lattice

Towards Autonomous MAV Exploration in Cluttered Indoor and Outdoor Environments
Korbinian Schmid
DLR, Robotics and Mechatronics Center (RMC)
Perception and Cognition Mobile Robots (XRotor Group)

Motivation
- No external navigation aids (GNSS)
- No reliable (high bandwidth, low latency) radio link
- Full on-board navigation solution
Towards Autonomous MAV Exploration in Cluttered Indoor and Outdoor Environments

Challenges

- Limited payload
- Limited computational resources
- Delayed data processing
- High system dynamics, inherently unstable
- High data load from exteroceptive sensors
- Computationally complex algorithms (mapping, path planning...)

Why Multicopter MAVs?

- Small
- Light-weight
- Agile
- Safe
- Cheap
- Easy to fly
- But: limited payload!

System architecture

Navigation Box Hardware

- 2 x 33 g
- 739 g
- 22 g
- 95 g
- 444 g

System architecture

Depth image calculation on FPGA

- Semi Global Matching (SGM) [Hirschmüller, 2008]
- FPGA implementation: [Gagrig et al., 2009]
  - acceleration by parallelization
  - acceleration by pipelining
  - 0.5 MPixel depth images at 14.6 Hz
INS filter design

- Synchronization of realtime and non realtime modules by sensor hardware trigger
- Direct system state:
  \[ x = \left[ \begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \right] \in \mathbb{R}^n \]
- High rate calculation by “Strap Down Algorithm” (SDA)
- Indirect system state:
  \[ \hat{x} = \left[ \begin{array}{c} \hat{x}_1 \\ \hat{x}_2 \\ \vdots \\ \hat{x}_n \end{array} \right] \in \mathbb{R}^n \]
- Estimation by indirect Extended Kalman Filter (EKF)
- Measurement delay compensation by filter state augmentation

Basic (direct) INS EKF with global position updates

- Prediction (state propagation):
  \[ \hat{x}_{k+1} = H \hat{x}_k + B_k \omega_k \]
  \[ P_{k+1} = \Phi P_k \Phi^T + \Gamma Q_k \Gamma^T \]
- Update:
  \[ \hat{x}_k = \Phi \hat{x}_k + \nu_k \]
  \[ P_{k} = (I - K_k H_k) P_{k+1} \]
  \[ K_k = P_{k}^T H_k^T (H_k P_{k+1} H_k^T + R_k)^{-1} \]
  \[ \hat{x}_k = \hat{x}_k + K_k (y_k - H_k \hat{x}_k) \]

State augmentation (stochastic cloning)

- General measurement:
  \[ \hat{x}_k = \hat{x}_k + \nu_k \]
- Odometry measurement (direct):
  \[ \hat{x}_k = \hat{x}_k + \nu_k = H_k (\hat{x}_k) + \nu_k \]
- “Stochastic cloning” [Koumeliotis, 2002]
- Augmentation of local position

Time:
\[ n_k \]
\[ n_k+1 \]
\[ n_k+2 \]
\[ n_{k+2} \]
\[ \hat{x}_{k+2} = \hat{x}_{k+2} + \nu_{k+2} \]
\[ k_{k+2} = k_{k+2} + \nu_{k+2} \]

INS EKF with relative time delayed measurements

- EKF:
  - State augmentation at exact time of measurement:
  - Saving direct system state
  - Cloning of indirect filter state
  - Prediction step as basic INS EKF
- Update:
  - Calculate delta pose from saved direct states
  - Calculate error residual from measurement
  - Standard EKF update referencing cloned indirect states in filter
  - Correction of direct states
- Instant processing of arriving (time delayed) measurements
Key frame based stereo odometry

- Delta measurements referencing key frames
- Locally drift free system state estimation
- EKF position SLAM with time delay compensation

Simulation (Trajectory)

Simulation (velocity, attitude)

- Velocity up to 4m/s
- Roll/Pitch angles up to 50deg

Simulation: FPGA acceleration vs. RMSE

- RMSE for delay variation:
  - const for position
  - linear for velocity

- RMSE for frequency variation:
  - exponential for position
  - exponential for velocity

Estimator properties fit well acceleration by FPGA

Robustness test

- 70 m trajectory
- Ground truth by tachymeter
- 5 s forced vision drop out with translational motion
- 1 s forced vision drop out with rotational motion
- Estimation error < 1.2 m
- Odometry error < 25.9 m
- Results comparable to runs without vision drop outs

Mixed indoor/outdoor exploration

- Autonomous indoor/outdoor flight of 60m
- Mapping resolution: 0.1m
- Leaving through a window
- Returning through door
Conclusion

- Multicopters for SAR and disaster management scenarios
- System concept for autonomous MAVs:
  - FPGA based stereo image processing
  - Key frame based stereo odometry
  - INS fusion with delay compensation by EKF
  - Mapping, path planning, mission control
- System state estimation improvement by FPGA acceleration
- Robust navigation concept for indoor/outdoor exploration

What's next?

Multicopters in SAR and disaster management scenarios

The RMC XRotor team

Thank you for your attention! Questions?

http://mobilerobots.dlr.de/systems/multicopters.
Or google: DLR XRotor

Visual Navigation for Flying Robots
Exploration, Multi-Robot Coordination and Coverage

Dr. Jürgen Sturm
Agenda for Today

- Exploration with a single robot
- Coordinated exploration with a team of robots
- Coverage planning
- Benchmarking

Mission Planning

- **Goal:** Generate and execute a plan to accomplish a certain (navigation) task
- **Example tasks**
  - Exploration
  - Coverage
  - Surveillance
  - Tracking
  - ...

Exploration and SLAM

- **SLAM is typically passive**, because it consumes incoming sensor data
- **Exploration actively guides the robot** to cover the environment with its sensors
- Exploration in combination with SLAM: Acting under pose and map uncertainty
- Uncertainty should/needs to be taken into account when selecting an action

Exploration

- By reasoning about control, the mapping process can be made much more effective
- **Question:** Where to move next?

This is also called the **next-best-view problem**
**Exploration**

- Choose the action that maximizes utility
  \[ a^* = \arg\max_{a \in A} U(m, a) \]
- **Question:** How can we define utility?

**Maximizing the Information Gain**

- Pick the action \( a \) that maximizes the information gain given a map \( m \)
  \[ a^* = \arg\max_{a \in A} IG(m, a) \]

**Information Theory**

- **Entropy** is a general measure for the uncertainty of a probability distribution
- Entropy = Expected amount of information needed to encode an outcome \( X = x \)
  \[ H(X) = E(I(X)) = E(-\log p(X)) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \]

**Example**

- Where should the robot go next?

**Maximizing the Information Gain**

- Pick the action \( a \) that maximizes the information gain given a map \( m \)
  \[ a^* = \arg\max_{a \in A} IG(m, a) \]

**Example: Binary Random Variable**

- Binary random variable \( X \in \{0, 1\} \)
- Probability distribution \( P(X = 1) = p \)
- How many bits do we need to transmit one sample of \( p(X) \)?
  - For \( p=0? \)
  - For \( p=0.5? \)
  - For \( p=1? \)
Example: Binary Random Variable

- Binary random variable $X \in \{0, 1\}$
- Probability distribution $P(X = 1) = p$
- How many bits do we need to transmit one sample of $p(X)$?
- Answer:

![Binary Random Variable Graph]

Information Theory

- Entropy of a grid map
  $$H(p(x_t)) = - \sum_{c \in \text{grid cells}} p(c) \log p(c) + (1 - p(c)) \log(1 - p(c))$$
- Information gain = reduction in entropy
  $$IG(t + 1 | t) = H(p(x_t)) - H(p(x_{t+1}))$$

Maximizing the Information Gain

- To compute the information gain one needs to know the observations obtained when carrying out an action
- This quantity is not known! Reason about potential measurements
  $$a^* = \arg \max_{a \in A} IG(m, a)$$
  $$a^* = \arg \max_{a \in A} \int IG(m, z)p(z | a)dz$$

Example: Map Entropy

![Map Entropy Diagram]

The overall entropy is the sum of the individual entropy values

Exploration Costs

- So far, we did not consider the cost of executing an action (e.g., time, energy, ...)
- Utility = uncertainty reduction – cost
- Select the action with the highest expected utility
  $$a^* = \arg \max_{a \in A} IG(m, a) - \alpha \cdot E(\text{cost}(m, a))$$
Exploration

- For each location $<x,y>$
  - Estimate the number of cells robot can sense (e.g., simulate laser beams using current map)
  - Estimate the cost of getting there

Greedy strategy: Select the candidate location with the highest utility, then repeat...

Exploration Actions

- So far, we only considered reduction in map uncertainty
- In general, there are many sources of uncertainty that can be reduced by exploration
  - Map uncertainty (visit unexplored areas)
  - Trajectory uncertainty (loop closing)
  - Localization uncertainty (active re-localization by re-visiting known locations)

Example: Active Loop Closing

- Reduce map uncertainty

- Reduce map + path uncertainty

Example: Active Loop Closing

- Entropy evolution
Example: Reduce uncertainty in map, path, and pose [Stachniss et al., 2005]

Selected target location

Dr. Jürgen Sturm, Computer Vision Group, TUM

Corridor Exploration [Stachniss et al., 2005]

- The decision-theoretic approach leads to intuitive behaviors: “re-localize before getting lost”
- Some animals show a similar behavior

Dr. Jürgen Sturm, Computer Vision Group, TUM

Multi-Robot Exploration

**Given:** Team of robots with communication

**Goal:** Explore the environment as fast as possible

Dr. Jürgen Sturm, Computer Vision Group, TUM

Complexity

- Single-robot exploration in known, graph-like environments is in general NP-hard
- Proof: Reduce traveling salesman problem to exploration
- Complexity of multi-robot exploration is exponential in the number of robots

Dr. Jürgen Sturm, Computer Vision Group, TUM

Motivation: Why Coordinate?

**Robot 1**

**Robot 2**

- Without coordination, two robots might choose the same exploration frontier

Dr. Jürgen Sturm, Computer Vision Group, TUM

Levels of Coordination

1. No exchange of information
2. Implicit coordination: Sharing a joint map
   - Communication of the individual maps and poses
   - Central mapping system
3. Explicit coordination: Determine better target locations to distribute the robots
   - Central planner for target point assignment
   - Minimize expected path cost / information gain / …
Typical Trajectories

Implicit coordination:

Explicit coordination:

Coordination Algorithm

In each time step:
- Determine set of exploration targets
  \( S = \{ s_1, \ldots, s_n \} \)
- Compute for each robot \( i \) and each target \( j \) the expected cost/utility \( C_{ij} \)
- Assign robots to targets using the Hungarian algorithm

Hungarian Algorithm

[Kuhn, 1955]
- Combinatorial optimization algorithm
- Solves the assignment problem in polynomial time \( O(n^3) \)
- General idea: Algorithm modifies the cost matrix until there is zero cost assignment

Hungarian Algorithm: Example

1. Compute the cost matrix (non-negative)

 Hungarian Algorithm: Example

2. Find minimum element in each row
Hungarian Algorithm: Example

3. Subtract minimum from each row element

4. Find minimum element in each column

5. Subtract minimum from each column element

6a. Assign (if possible)

6b. If no assignment is possible:
- Connect all 0’s by vertical/horizontal lines
- Find the minimum in all remaining elements and subtract
- Add to all double crossed out coefficients
- Repeat step 2 – 6

If there are not enough targets:
Copy targets to allow multiple assignments
Example: Segmentation-based Exploration

[Wurm et al., IROS 2008]

- Two-layer hierarchical role assignments using Hungarian algorithm (1: rooms, 2: targets in room)
- Reduces exploration time and risk of interferences

Summary: Exploration

- Exploration aims at generating robot motions so that an **optimal map** is obtained
- **Coordination** reduces exploration time
- **Hungarian algorithm** efficiently solves the assignment problem (centralized, 1-step lookahead)
- Challenges (active research):
  - Limited bandwidth and **unreliable communication**
  - **Decentralized planning** and task assignment

Coverage Path Planning

**Given:** Known environment with obstacles

**Wanted:** The shortest trajectory that ensures complete (sensor) coverage

Coverage Path Planning: Applications

- For flying robots
  - Search and rescue
  - Area surveillance
  - Environmental inspection
  - Inspection of buildings (bridges)
- For service robots
  - Lawn mowing
  - Vacuum cleaning
- For manipulation robots
  - Painting
  - Automated farming
- For flying robots

Coverage Path Planning

- What is a good coverage strategy?
- What would be a good cost function?
Coverage Path Planning

- What is a good coverage strategy?
- What would be a good cost function?
  - Amount of redundant traversals
  - Number of stops and rotations
  - Execution time
  - Energy consumption
  - Robustness
  - Probability of success
  - ...

- Related to the traveling salesman problem (TSP):
  “Given a weighted graph, compute a path that visits every vertex once”
- In general NP-complete
- Many approximations exist
- Many approximate (and exact) solvers exist

Coverage of Simple Shapes

- Approximately optimal solution often easy to compute for simple shapes (e.g., trapezoids)

Idea

[Mannadiar and Rekleitis, ICRA 2011]
Coverage Based On Cell Decomposition

[Mannadiar and Rekleitis, ICRA 2011]

Approach:
1. Decompose map into “simple” cells
2. Compute connectivity between cells and build graph
3. Solve coverage problem on reduced graph

Step 1: Boustrophedon Cellular Decomposition

[Mannadiar and Rekleitis, ICRA 2011]

- Similar to trapezoidal decomposition
- Can be computed efficiently in 2D

Step 2: Build Reeb Graph

[Mannadiar and Rekleitis, ICRA 2011]

- Vertices = Critical points (that triggered the split)
- Edges = Connectivity between critical points

Step 3: Compute Euler Tour

[Mannadiar and Rekleitis, ICRA 2011]

- Extend graph so that vertices have even order
- Compute Euler tour (linear time)

Resulting Coverage Plan

[Mannadiar and Rekleitis, ICRA 2011]

- Follow the Euler tour
- Use simple coverage strategy for cells
- Note: Cells are visited once or twice

What are the desired properties of a good scientific experiment?

- Reproducibility / repeatability
  - Document the experimental setup
  - Choose (and motivate) an evaluation criterion
- Experiments should allow you to validate/falsify competing hypotheses

Current trends:
- Make data available for review and criticism
- Same for software (open source)
**Benchmarks**

- Effective and affordable way of conducting experiments
- Sample of a task domain
- Well-defined performance measurements
- Widely used in computer vision and robotics
- **Which benchmark problems do you know?**

**Example Benchmark Problems**

**Computer Vision**

- Middlebury datasets (optical flow, stereo, ...)
- Caltech-101, PASCAL (object recognition)
- Stanford bunny (3d reconstruction)

**Robotics**

- RoboCup competitions (robotic soccer)
- DARPA challenges (autonomous car)
- SLAM datasets

---

**Image Denoising: Lenna Image**

- 512x512 pixel standard image for image compression and denoising
- Lena Söderberg, Playboy magazine Nov. 1972
- Scanned by Alex Sawchuck at USC in a hurry for a conference paper

http://www.cs.cmu.edu/~chuck/lennapg/  

---

**Object Recognition: Caltech-101**

- Pictures of objects belonging to 101 categories
- About 40-800 images per category
- Recognition, classification, categorization

---

**RoboCup Initiative**

- Evaluation of full system performance
- Includes perception, planning, control, ...
- Easy to understand, high publicity
- “By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, complying with the official rule of the FIFA, against the winner of the most recent World Cup.”
### SLAM Evaluation

- **Intel dataset:** laser + odometry [Haehnel, 2004]
- **New College dataset:** stereo + omni-directional vision + laser + IMU [Smith et al., 2009]
- **TUM RGB-D dataset** [Sturm et al., 2011/12]

### TUM RGB-D Dataset

- RGB-D dataset with ground truth for SLAM evaluation
- Two error metrics proposed (relative and absolute error)
- Online + offline evaluation tools
- Training datasets (fully available)
- Validation datasets (ground truth not publicly available to avoid overfitting)

### Recorded Scenes

- Various scenes (handheld/robot-mounted, office, industrial hall, dynamic objects, ...)
- Large variations in camera speed, camera motion, illumination, environment size, ...

### Dataset Acquisition

- Motion capture system
  - Camera pose (100 Hz)
- Microsoft Kinect
  - Color images (30 Hz)
  - Depth maps (30 Hz)
  - IMU (500 Hz)
- External video camera (for documentation)

### Motion Capture System

- 9 high-speed cameras mounted in room
- Cameras have active illumination and pre-process image (thresholding)
- Cameras track positions of retro-reflective markers

### Calibration

Calibration of the overall system is not trivial:
1. Mocap calibration
2. Kinect-mocap calibration
3. Time synchronization
Calibration Step 1: Mocap

- Need at least 2 cameras for position fix
- Need at least 3 markers on object for full pose
- Calibration stick for extrinsic calibration

Example: Raw Image from Mocap

Example: Position Triangulation of a Single Marker

Example: Tracked Object (4 Markers)

Example: Recorded Trajectory
Calibration Step 2: Mocap-Kinect

- Need to find transformation between the markers on the Kinect and the optical center
- Special calibration board visible both by Kinect and mocap system (manually gauged)

Calibration Step 3: Time Synchronization

- Assume a constant time delay between mocap and Kinect messages
- Choose time delay that minimizes reprojection error during checkerboard calibration

Dataset Website

- In total: 39 sequences (19 with ground truth)
- One ZIP archive per sequence, containing
  - Color and depth images (PNG)
  - Accelerometer data (timestamp ax ay az)
  - Trajectory file (timestamp tx ty qx qy qz qw)
- Sequences also available as ROS bag and MRPT rawlog

http://vision.in.tum.de/data/datasets/rgbd-dataset
**What Is a Good Evaluation Metric?**

- Compare camera trajectories
  - Ground truth trajectory $Q_1, \ldots, Q_n \in \text{SE}(3)$
  - Estimate camera trajectory $P_1, \ldots, P_n \in \text{SE}(3)$
- Two common evaluation metrics
  - Relative pose error (drift per second)
  - Absolute trajectory error (global consistency)

**Relative Pose Error (RPE)**

- Measures the (relative) drift
- Recommended for the evaluation of visual odometry approaches

$$E_i := \left( Q_i^{-1} Q_{i+\Delta} \right)^{-1} \left( P_i^{-1} P_{i+\Delta} \right)$$

**Absolute Trajectory Error (ATE)**

- Measures the global error
- Requires pre-aligned trajectories
- Recommended for SLAM evaluation

$$E_i := Q_i^{-1} S P_i$$

**Evaluation metrics**

- Average over all time steps
- Reference implementations for both evaluation metrics available
- Output: RMSE, Mean, Median (as text)
- Plot (png/pdf, optional)
**Summary – TUM RGB-D Benchmark**

- Dataset for the evaluation of RGB-D SLAM systems
- Ground-truth camera poses
- Evaluation metrics + tools available

**Discussion on Benchmarks**

**Pro:**
- Provide objective measure
- Simplify empirical evaluation
- Stimulate comparison

**Con:**
- Introduce bias towards approaches that perform well on the benchmark (overfitting)
- Evaluation metrics are not unique (many alternative metrics exist, choice is subjective)

**Lessons Learned Today**

- How to generate plans that are robust to uncertainty in sensing and locomotion
- How to explore an unknown environment
  - With a single robot
  - With a team of robots
- How to generate plans that fully cover known environments
- How to benchmark SLAM algorithms